**CURSO 2018-2019**
DATA SCIENCE MASTER
UNIVERSIDAD DE GRANADA

# Aplicaciones de Ciencias de Datos y Tecnologías Inteligentes: Técnicas automáticas de inversión bursátil.

Carlos Manuel Sequí Sánchez

13 de junio de 2019

## Índice

In this workflow I'm going to create an algorithm which will make use of stock market techniques in order to decide when to buy or to sell stocks in stock market. I've downloaded Adida's data from years 2017 to 2019 which has 504 instances/days candlesticks information. This dataset is available in Yahoo finance.

## 1. Preprocessing dataset.

### Loading libraries:

```python
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

### Reading dataset

```python
data = pd.read_csv("E:\MASTER\Aplicaciones para la ciencia de datos\
    Inversion en bolsa\ADDYY.csv")
```

### Deleting unnecessary columns

```python
data = data.drop(["Date","Adj Close","Volume"],axis=1)
print(data)
```

## 2. Calculating fuzzy inputs.

Let's extract fuzzy imputs from data to detect bullish patterns:

- Lupper, Llower and Lbody: lenght of the upper shadow, lower shadow and body respectively.
- Trend: the trend of the last two candlesticks using the difference in percentage between their close values in the consecutive previous sessions: t, t-1 and t-2.

```python
data['Lupper'] = 100 * ((data['High']-data[['Open','Close']].max(axis=1))/
    data['Open'])
data['Llower'] = 100 * ((data[['Open','Close']].min(axis=1) - data['Low'])/
    data['Open'])
data['Lbody'] = 100 * ((data['Close'] - data['Open'])/data['Close'])
data['Trend'] = 0 # initializing trend
for i in range(1,len(data)):
data['Trend'][i] = 100 * ((data['Close'][i] - data['Close'][i-1])/data['
    Close'][i])
```

## 3. Creation of classification approaches.

Let's create an 'action' attribute based on our own different decisions to buy or sell stocks, helping us with the information that gives us the fuzzy output bullish, which represents the prediction of an upward trend.
These are the rules/patterns we're going to use to make actions:

- Simple base approach.
- Detecting Kicking bullish approach. (fuzzy)
- Detecting Hammer approach. (fuzzy)

## 3.1. Simple base approach

This code creates a column with buy or sell operations based only on the difference between open and close values: when it is positive we must sell, and viceversa.

```
data['action'] = np.where((data['Close']-data['Open'])>0, 'SELL', 'BUY')
```

## 3.2. Detecting Kicking Bullish approach

Let's add an approach based on detecting Kicking Bullish: we buy if we detect Kicking Bullish. The confidence on this pattern greatly depends on the size of the gap between the strong black cnadlestick (or a black Marubozu) of the first day and the white candlestick (a white Marubozu) of the next day.

```
for i in range(1,len(data)):
if data['High'][i-1] == data['Open'][i-1] and \
data['Low'][i-1] == data['Close'][i-1] and \
data['High'][i] == data['Open'][i] and \
data['Low'][i] == data['Close'][i] and \
data['Low'][i] > data['High'][i-1]:
data['action'][i] = 'BUY'
```

## 3.3. Detecting Hammer approach.

Let's add the last approach based on detecting Hammer: we buy if we detect a Hammer. The reliability of the Hammer pattern is based on the relative length of the lower shadow of the candlestick with respect to its body; the greater, the better. The color of the body is not important. In addition, the strength of the latest trend has been included, since traders consider that when the bearish trend recedes, it could be interpreted as a resistance to continue the current trend, and therefore, it means a turning point in the trend.

```
for i in range(2,len(data)):
if data['Trend'][i] < 0 and \
data['Trend'][i-1] < 0 and \
data['Trend'][i-2] < 0 and \
data['Low'][i] < data['Low'][i-1] and \
((data['High'][i] == data[['Open','Close']].max(axis=1)[i]) or ((data['High'][i]-data[['Open','Close']].max(axis=1)[i]) < (data['Lbody'][i]/5))) and \
(data[['Open','Close']].min(axis=1)[i]-data['Low'][i]) > (2*abs(data['Open'][i]-data['Close'][i])):
data['action'][i] = 'BUY'
```

## 4. Creation of train and test datasets.

We get the train data sets, selecting the first 67 % of the dataset in order to respect the time series:

```
trainSet = data.head(int(len(data)*0.67))
X_train = trainSet.drop(["action"],axis=1)
y_train = trainSet['action']
```

We get the test data sets with the lasting 33 % of the dataset:

```
testSet = data.tail(int(len(data)*0.33))
X_test = testSet.drop(["action"],axis=1)
y_test = testSet['action']
```

## 5. Classification: KNN

Creating the model:

```
neigh = KNeighborsClassifier(n_neighbors=3)
```

Fitting the model...

```
neigh.fit(X_train, y_train)
```

Getting predictions...

```
pred = neigh.predict(X_test)
```

Results:

```
print(accuracy_score(y_test, pred))
```

0.7228915662650602
And this is our model's accuracy score using the techniques described and a KNN model to predict actions.