TP Algorithmic Trading : Implementing Trading Strategies

Adnane Bazouz, Dieye Souleymane

1. Moving Average Crossover Trading Strategy

In [1]:

```
# decomment lines below to install packages if non existent

#! pip install seaborn
#! pip install statsmodels
```

In [2]:

```
# importing relevant libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn
import statsmodels.api as sm
```

In this section, we implement the Moving Average Crossover Trading Strategy. In this strategy we compare two quantities : the short term simple moving average SMA (sma_ct) and the long term SMA (sma_lt). Our strategy is the following :

```
- If sma_ct > sma_lt we buy the stock, our position is then +1
- If sma_ct < sma_lt we sell the stock, our position is then -1
```

In [32]:

```
# Reading data
tr_eikon_path = '~/Desktop/AlgoTrading/tr_eikon_eod_data.csv'
df_tr_eikon = pd.read_csv(tr_eikon_path, index_col = 0, parse_dates = True).dropna()
df_tr_eikon.head()
```

Out[32]:

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	.SPX	.VIX	EUR=	XAU=	G
Date											
2010-01-04	30.572827	30.950	20.88	133.90	173.08	113.33	1132.99	20.04	1.4411	1120.00	47
2010-01-05	30.625684	30.960	20.87	134.69	176.14	113.63	1136.52	19.35	1.4368	1118.65	48
2010-01-06	30.138541	30.770	20.80	132.25	174.26	113.71	1137.14	19.16	1.4412	1138.50	49
2010-01-07	30.082827	30.452	20.60	130.00	177.67	114.19	1141.69	19.06	1.4318	1131.90	49
2010-01-08	30.282827	30.660	20.83	133.52	174.31	114.57	1144.98	18.13	1.4412	1136.10	49

In [4]:

```
# Here we implement a function that gives the return given values for sma and lma windows

def get_strategy_return(s_window, l_window, dataframe, symbol = "AAPL.O", verbose = True) :

    """This function computes the return of the Moving Average Cross Over Strategy

    Parameters :
        - s_window : the short term moving average window (int)
        - l_window : the long term mmoving average window (int)
        - dataframe : the dataframe of stock prices

    Returns : (float) The profit of the strategy
    """

    # long term and short term sma
    sma_ct_vec = dataframe.rolling(window = s_window, center = False).mean()
    sma_lt_vec = dataframe.rolling(window = l_window, center = False).mean()

    # Here we get our positions according to our strategy
    positions_ = sma_ct_vec > sma_lt_vec

    # We map the vector from {0,1} to {-1,1}
    positions_ = 2*positions_ - 1

    # Computing profit
    log_vals = np.log(dataframe[symbol])
    returns_vec_ = log_vals.diff()
    cum_returns = returns_vec_.cumsum()
    profits_ = np.multiply(positions_[symbol], returns_vec_)
    cum_profits = profits_.cumsum()

    expected_prof = cum_profits[-1]

    if verbose :
        print ('Profit for short term window %d and long term window %d is : %.4f'%(s_windo

    return expected_prof, cum_profits, positions_[symbol], cum_returns
```

In this section we backtest the strategy using the values 42 and 252 for short moving average and long moving average respectively. And show the returns it provides.

In [5]:

```
s_window = 42
l_window = 252
dataframe = df_tr_eikon
```

In [6]:

```
return_42_252, _, _, _ = get_strategy_return(s_window, l_window, dataframe )
```

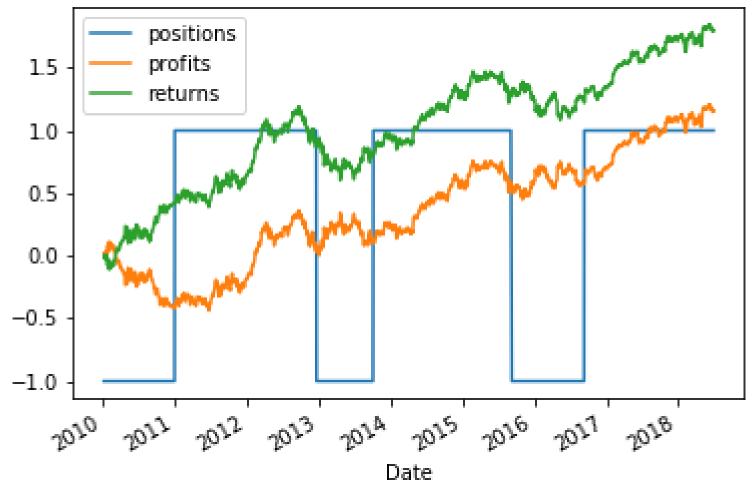
```
Profit for short term window 42 and long term window 252 is : 1.1595
```

In [7]:

```
_, cum_profits, positions_, cum_returns = get_strategy_return(s_window, l_window, dataframe

#ax = df.plot(kind='bar')
#ax.legend(["AAA", "BBB"]);
px = positions_.plot()
cx = cum_profits.plot()
rx = cum_returns.plot()
px.legend(["positions", "profits", "returns"])
```

Out[7]:

```
<matplotlib.legend.Legend at 0x1d6c2233a90>
```



OPTIMIZING PARAMETERS

The values 42 and 252 were chosen randomly. Now we run multiple backtests, varying the moving average windows each time, to find the values with the highest return.

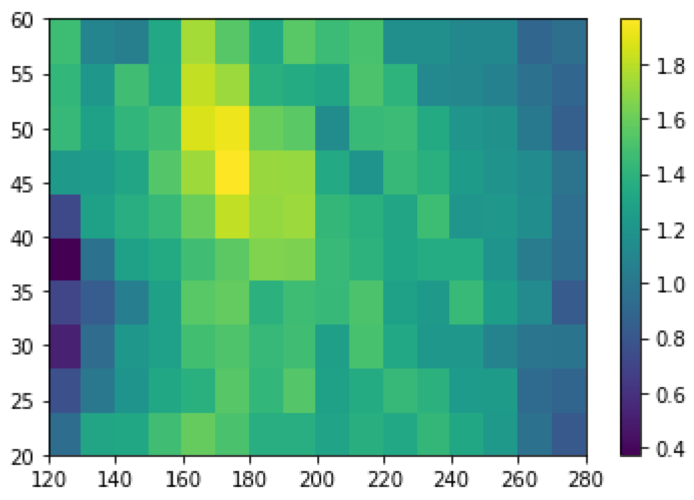
In [8]:

```
# We create vectors of values of short and long sma
short_sma_window = np.arange(20, 61, 4)
long_sma_window = np.arange(120, 281, 10)

# for each (short_sma, long_sma) we get the return and save it in return matrix
return_matrix = np.zeros((len(short_sma_window), len(long_sma_window)))

print ("Computing returns...\n")
for p, s_wind in enumerate(short_sma_window) :

    for q, l_wind in enumerate(long_sma_window) :

        return_matrix[p, q], _, _, _ = get_strategy_return(int(s_wind), int(l_wind), datafr


smax_position, lmax_position = np.where(return_matrix == return_matrix.max())

print ("Maximum return is obtained for short term window %d and long term window %d, and is
                                                                   long_sma_window[lmax_pos
                                                                  return_matrix.max()))
```

```
Computing returns...

Maximum return is obtained for short term window 44 and long term window 17
0, and is 1.9637
```

In [9]:

```
# we plot the heat map showing the results
plt.pcolor(long_sma_window, short_sma_window, return_matrix )
plt.colorbar()
plt.show()
```

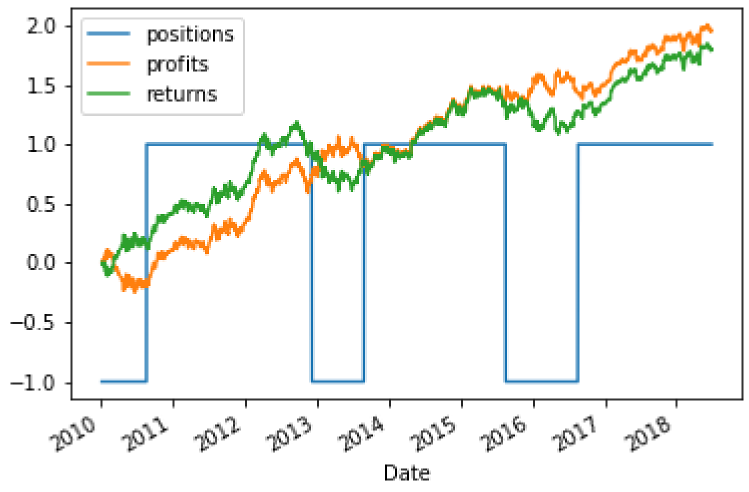


In [10]:

```
_, cum_profits, positions_, cum_returns = get_strategy_return(50, 160, dataframe, verbose =

#ax = df.plot(kind='bar')
#ax.legend(["AAA", "BBB"]);
px = positions_.plot()
cx = cum_profits.plot()
rx = cum_returns.plot()
px.legend(["positions", "profits", "returns"])
```

Out[10]:

```
<matplotlib.legend.Legend at 0x1d6c25ac4e0>
```



2. Fibonacci Trading Strategies

Now we examine a strategy based on the Fibonacci sequence. Fibonacci Retracements are ratios used to identify potential reversal levels. The most popular Fibonacci Retracements are 61.8% and 38.2%

These ratios, 23.6%, 38.2%, and 61.8%, are widely used for time series analysis to find support level. Whenever the price moves substantially upwards or downwards, it usually tends to retrace back before it continues to move in the original direction. For example, if the stock price has moved from €200 to €250, then it is likely to retrace back to €230 before it continues to move upward. The retracement level of €230 is forecasted using the Fibonacci ratios.

We can arrive at €230 by :

Total up move = €250 – €200 = €50

38.2% of up move = 38.2% * 50 = €19.1

Retracement forecast = €250 – $19.1 = €230.9

Any price level below €230 provides a good opportunity for the traders to enter into new positions in the direction of the trend. Likewise, we can calculate for the other Fibonacci ratios.

2.1 Finidng Fibonacci retracement levels

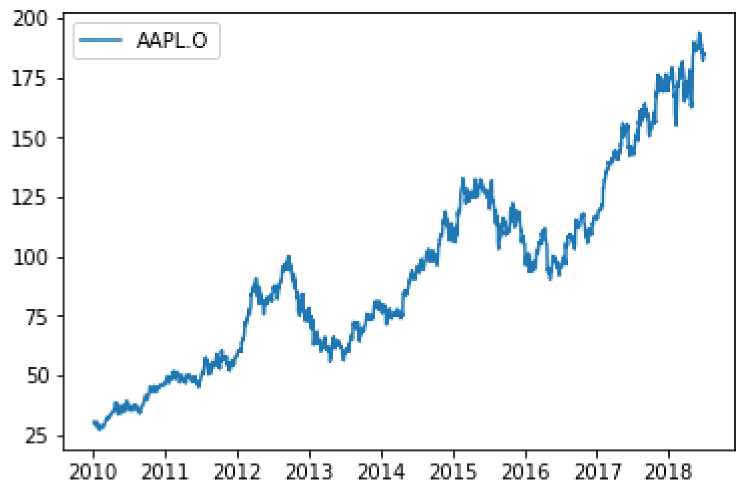
Retracements are the price movements that go against the original trend. To forecast the Fibonacci retracement level we should first identify the total up move or total down move. To mark the move, we need to pick the most recent high and low on the chart.

In [11]:

```
symbol = 'AAPL.O'
figures, axes = plt.subplots()
stock_prices = df_tr_eikon[symbol]
axes.plot(stock_prices)
axes.legend([symbol])
```

Out[11]:

```
<matplotlib.legend.Legend at 0x1d6c27e26d8>
```



In [12]:

```
# Get min and max prices
price_min = stock_prices.min()
price_max = stock_prices.max()
print ("Min price is %.4f"%price_min)
print ("Max price is %.4f"%price_max)
```

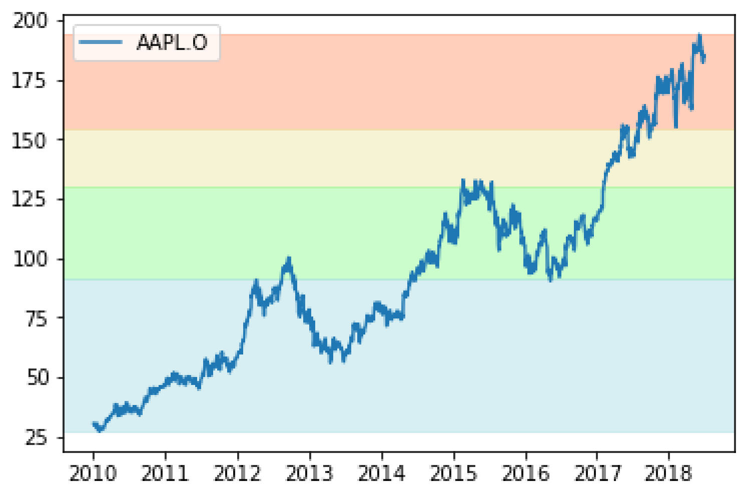
```
Min price is 27.4357
Max price is 193.9800
```

Now we compute retracement levels for Fibonacci ratios of 23.6%, 38.2% and 61.8% :

In [13]:

```
# Fibonacci Levels considering original trend as upward move
spread = price_max - price_min
level1 = price_max - 0.236 * spread
level2 = price_max - 0.382 * spread
level3 = price_max - 0.618 * spread

axes.axhspan(level1, price_max, alpha=0.5, color='lightsalmon')
axes.axhspan(level2, level1, alpha=0.5, color='palegoldenrod')
axes.axhspan(level3, level2, alpha=0.5, color='palegreen')
axes.axhspan(price_min, level3, alpha=0.5, color='powderblue')

figures
```

Out[13]:



In [14]:

```
def fibonacciRetracementStrategy(ratio, dataframe, symbol = 'AAPL.O') :

    """" This function computes the profit of the Fibonacci Retracement Strategy
    Parameters :
        - ratio (float) : Fibonacci ratio for the strategy
        - dataframe : a pandas DataFrame object with prices
    Returns profit of the strategy
    """
    prices = dataframe[symbol].dropna()

    # we get retracement levels as explained in intro
    dataframe['pastWave'] = prices.shift(1) - prices.shift(2)
    dataframe['potentialRetracementLevel'] = prices.shift(1) - ratio*dataframe['pastWave']
    dataframe['variation'] = prices.diff()



    wave_up = dataframe['pastWave'] > 0
    wave_down = dataframe['pastWave'] < 0

    is_decreasing = dataframe['variation'] < 0
    is_increasing = dataframe['variation'] > 0

    is_higher = prices > dataframe['potentialRetracementLevel']
    is_lower = prices < dataframe['potentialRetracementLevel']

    # condition 1 : (corrective wave)the wave moves up and the entire wave is below the lev
    condition1 = wave_up & is_decreasing & is_higher
    dataframe.loc[condition1, 'position'] = -1

    condition2 = wave_down & is_increasing & is_lower
    dataframe.loc[condition2, 'position'] = 1

    condition3 = (wave_up & is_decreasing & is_lower) | (wave_up&is_increasing)
    dataframe.loc[condition3, 'position'] = 1

    condition4 = (wave_down&is_increasing&is_higher) | (wave_down&is_decreasing)
    dataframe.loc[condition4, 'position'] = -1

    # returns
    log_vals_ = np.log(prices)
    returns_vec = log_vals_.diff()

    # profits
    positions = dataframe['position']
    profits = np.multiply(positions, returns_vec)
    dataframe['profit'] = profits

    cum_returns = returns_vec.cumsum()
    cum_profits = profits.cumsum()

    strategy_profit = profits.sum()

    return strategy_profit, cum_profits, cum_returns, positions, dataframe
```

In [15]:

```
dataframe = df_tr_eikon
ratios = [.236, .382, .618]

for ratio in ratios :
    profit, cum_profits_,cum_returns_, _, _= fibonacciRetracementStrategy(ratio, dataframe)
    print ("Profit for ratio %.3f is : %.4f"%(ratio, profit))
```

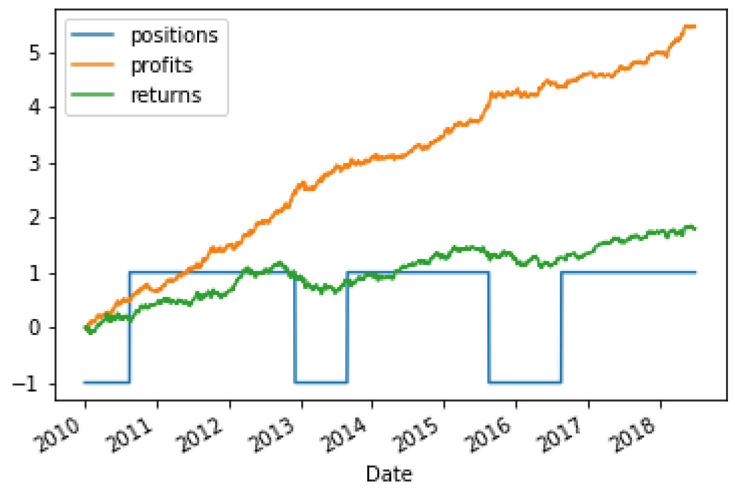
```
Profit for ratio 0.236 is : 2.5764
Profit for ratio 0.382 is : 3.5924
Profit for ratio 0.618 is : 5.4682
```

In [16]:

```
px = positions_.plot()
cx = cum_profits_.plot()
rx = cum_returns_.plot()
px.legend(["positions", "profits", "returns"])
```

Out[16]:

```
<matplotlib.legend.Legend at 0x1d6c2650c50>
```



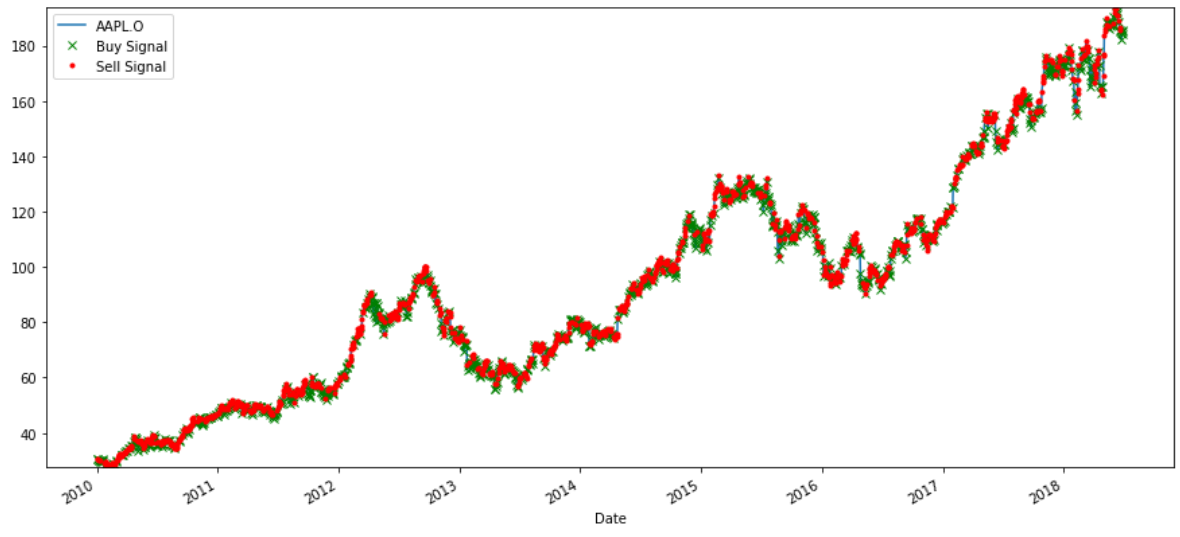
In [17]:

```
def seeOrders(ratio, symbol, df):

    plt.figure(figsize=(15,7))
    profit, _, _, _ ,dataframe = fibonacciRetracementStrategy(ratio, df)
    print("Profit : %.4f"%profit)

    dataframe[symbol].plot()
    buy = dataframe[symbol].copy()
    sell = dataframe[symbol].copy()
    buy[(((dataframe['pastWave'] < 0) & ((dataframe['variation'] > 0) & (dataframe[symbol]
    sell[(((dataframe['pastWave'] > 0) & ((dataframe['variation'] < 0) & (dataframe[symbol]

    buy.plot(color='g', linestyle='None', marker='x')
    sell.plot(color='r', linestyle='None', marker='.')

    x1,x2,y1,y2 = plt.axis()
    plt.axis((x1,x2,dataframe[symbol].min(),dataframe[symbol].max()))
    plt.legend([symbol, "Buy Signal", 'Sell Signal'])
    plt.show()
```

In [18]:

```
symbol = 'AAPL.O'
ratio = 0.618
seeOrders(ratio, symbol, df_tr_eikon)
```

```
Profit : 5.4682
```



Effect of ratio choice :

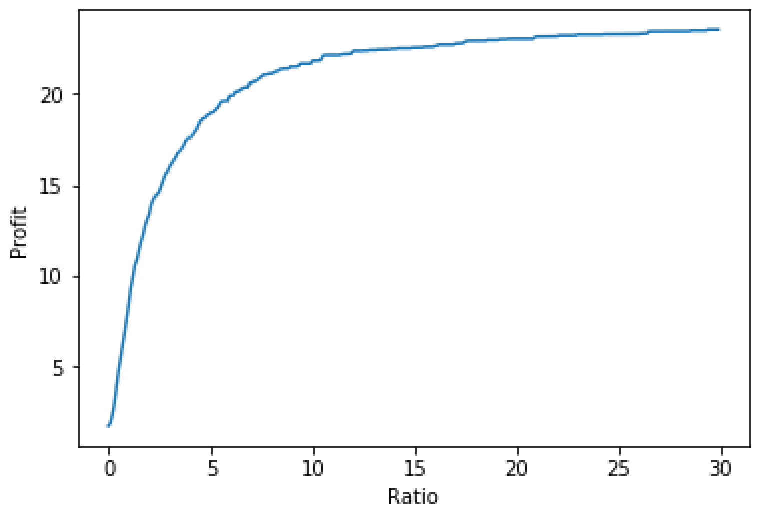
In [19]:

```
dataframe = df_tr_eikon
profits = []
ratios_ = np.arange(0, 30, .1)

print ("Computing profits for ratios varying from %.2f to %.2f ... "%(min(ratios_), max(rat
for ratio in ratios_ :

    profit, _, _, _, _ = fibonacciRetracementStrategy(ratio, dataframe)
    profits.append(profit)

print ("Profits computed ")
plt.plot(ratios_, profits)
plt.xlabel("Ratio")
plt.ylabel("Profit")
```

```
Computing profits for ratios varying from 0.00 to 29.90 ...
Profits computed
```

Out[19]:

```
Text(0,0.5,'Profit')
```



3. Pairs Trading Strategy

Definition:

Two series $X_t = (X_1(t), \dots, X_n(t))\ and\ Y_t = (Y_1(t), \dots, Y_n(t))$ are cointegrated if the ratio Y/X varies around a mean α

Principle of Pairs trading:

First of all, we have to find a cointegrated pairs. As our cointegrated pairs drift towards and apart from each other then we have a variation of the of the spread. In order to make a pairs trade, we have to buy one of the securities and sell the other one. This Strategy is a market neutral strategy, it allows to make profit from any market trend. Because:

- If both seurities move on the same direction, we neither gain or lose money
- If $Y/X > \alpha$, then we expect a decrease of the ratio Y/X . So we will sell X and buy Y
- If $Y/X < \alpha$, then we expect an increase of the ratio Y/X . So we will buy X and sell Y

Searching for Cointegrated Pairs :

In this part we are going to compute a statistical test between all the Pairs of our dataset in order to find the co-integrated pairs. We will define a threshold of 1% and we will consider only the pairs for which the P-value is under the threshold.

In [20]:

```
from statsmodels.tsa.stattools import coint

def coIntegratedPairsResearch(dataframe, threshold = 0.5):

    """ This function returns the cointegrated pairs of a given dataframe and the threshold

    shape_ = dataframe.shape[1]

    # we store cointegration values in the following matrix
    pValueMatrix = np.ones((shape_, shape_))
    indicesList = list(dataframe)

    coIntegratedPairsList = []
    for i in range(shape_) :
        for j in range(i+1, shape_) :
            col_i = dataframe[indicesList[i]]
            col_j = dataframe[indicesList[j]]
            cointegration = coint(col_i, col_j)
            pvalue = cointegration[1]
            pValueMatrix[i, j] = pvalue
            if pvalue < threshold:
                coIntegratedPairsList.append((indicesList[i], indicesList[j], pvalue))
    # we sort by pvalue
    coIntegratedPairsList.sort(key=lambda x:x[2])

    return pValueMatrix, coIntegratedPairsList
```

In [21]:

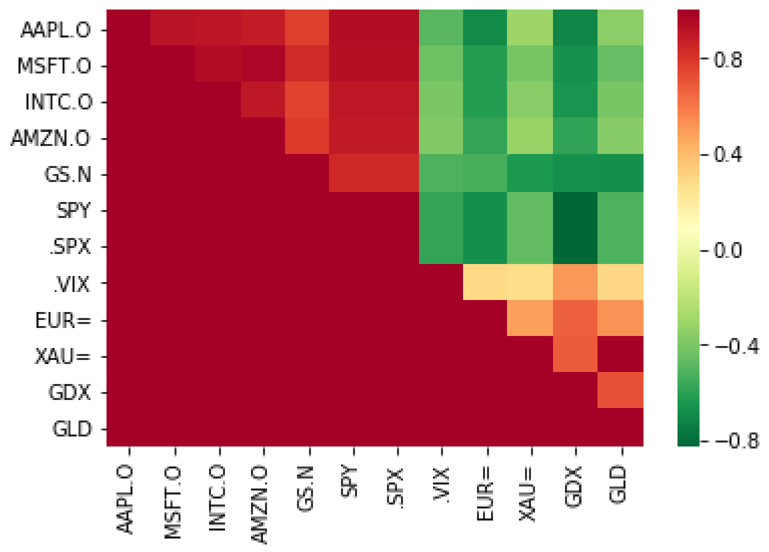
```
from scipy import stats

def correlationPairsResearch(dataframe, threshold = .9):

    """ This function returns the correlated pairs of a given dataframe and the threshold v
    n = dataframe.shape[1]
    correlationMatrix = np.ones((n, n))
    indicesList = list(dataframe)
    correlationPairsList = [] # initilize the list for cointegration
    for i in range(n):
        for j in range(i+1, n): # for j bigger than i
            pvalue = stats.pearsonr(dataframe[indicesList[i]], dataframe[indicesList[j]])[0
            correlationMatrix[i, j] = pvalue
            if abs(pvalue) > threshold :
                correlationPairsList.append((indicesList[i], indicesList[j], pvalue))
    correlationPairsList.sort(key=lambda x:-x[2])

    return correlationMatrix, correlationPairsList
```

In [22]:

```
tr_eikon_path = '~/Desktop/AlgoTrading/tr_eikon_eod_data.csv'
df = pd.read_csv(tr_eikon_path, index_col = 0, parse_dates = True).dropna()

correlationMatrix, correlationPairsList = correlationPairsResearch(df)
correlationMatrixDataFrame = pd.DataFrame(correlationMatrix)

coIntegrationMatrix, coIntegratedPairsList = coIntegratedPairsResearch(df)

seaborn.heatmap(correlationMatrixDataFrame, xticklabels=list(df),
                yticklabels=list(df), cmap='RdYlGn_r')
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d6c299c5c0>
```



In [23]:

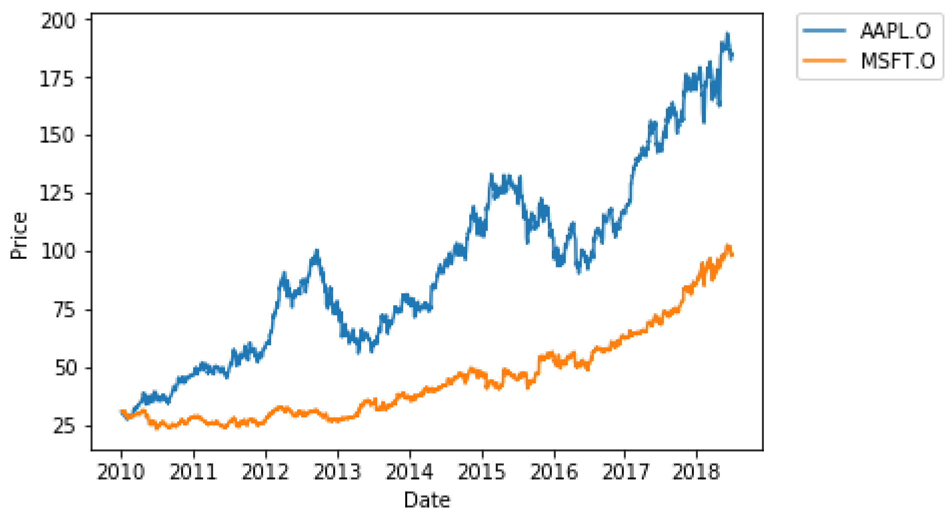
```
for pair in coIntegratedPairsList:
    print("Stock {} and stock {} has a co-integration score of {}".format(pair[0],pair[1],r
print("------------------------------------------------------------------------------------
for pair in correlationPairsList:
    print("Stock {} and stock {} has a correlation score of {}".format(pair[0],pair[1],rour
```

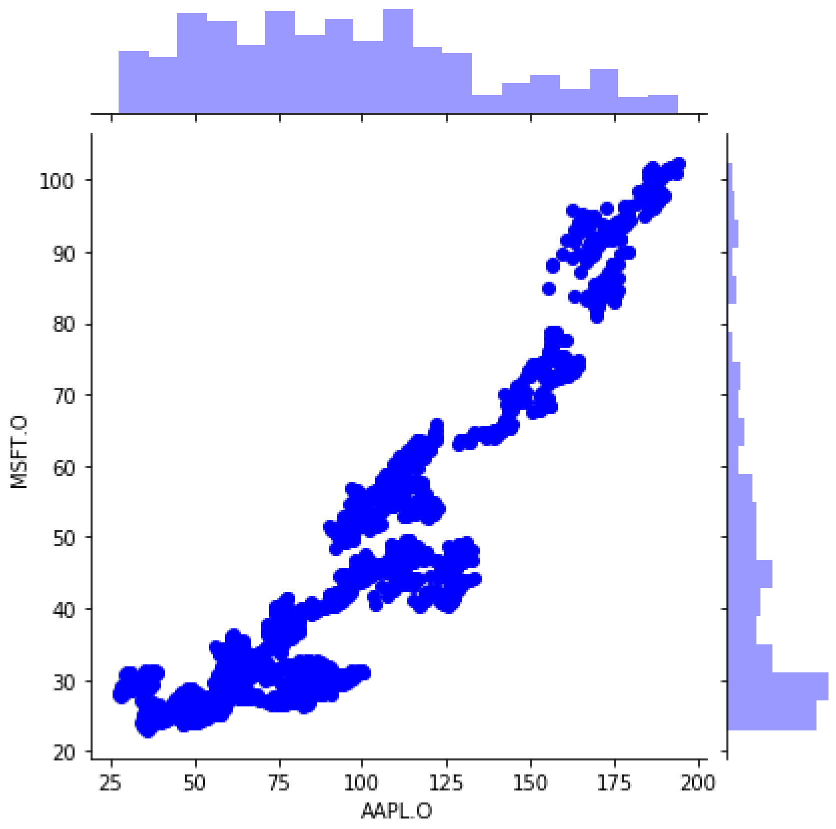
```
Stock SPY and stock .SPX has a co-integration score of 0.0
Stock .VIX and stock GDX has a co-integration score of 0.0006
Stock .VIX and stock XAU= has a co-integration score of 0.0016
Stock .VIX and stock EUR= has a co-integration score of 0.005
Stock .VIX and stock GLD has a co-integration score of 0.0081
Stock AAPL.O and stock INTC.O has a co-integration score of 0.0595
Stock MSFT.O and stock INTC.O has a co-integration score of 0.076
Stock INTC.O and stock AMZN.O has a co-integration score of 0.1045
Stock AAPL.O and stock MSFT.O has a co-integration score of 0.1798
Stock INTC.O and stock .SPX has a co-integration score of 0.1857
Stock EUR= and stock GLD has a co-integration score of 0.1883
Stock EUR= and stock XAU= has a co-integration score of 0.1959
Stock MSFT.O and stock AMZN.O has a co-integration score of 0.203
Stock XAU= and stock GDX has a co-integration score of 0.2143
Stock GS.N and stock SPY has a co-integration score of 0.2293
Stock GS.N and stock .SPX has a co-integration score of 0.2296
Stock INTC.O and stock SPY has a co-integration score of 0.2515
Stock GS.N and stock GLD has a co-integration score of 0.3471
Stock AAPL.O and stock .SPX has a co-integration score of 0.3917
Stock GS.N and stock XAU= has a co-integration score of 0.4066
Stock EUR= and stock GDX has a co-integration score of 0.4115
Stock GDX and stock GLD has a co-integration score of 0.4159
Stock AAPL.O and stock SPY has a co-integration score of 0.4335
Stock INTC.O and stock GS.N has a co-integration score of 0.4382
Stock GS.N and stock GDX has a co-integration score of 0.4916
----------------------------------------------------------------------------
-----------------------------
Stock SPY and stock .SPX has a correlation score of 1.0
Stock XAU= and stock GLD has a correlation score of 0.9982
Stock MSFT.O and stock AMZN.O has a correlation score of 0.976
Stock MSFT.O and stock INTC.O has a correlation score of 0.951
Stock AAPL.O and stock SPY has a correlation score of 0.9449
Stock AAPL.O and stock .SPX has a correlation score of 0.9448
Stock MSFT.O and stock .SPX has a correlation score of 0.9373
Stock MSFT.O and stock SPY has a correlation score of 0.937
Stock AAPL.O and stock MSFT.O has a correlation score of 0.9244
Stock AAPL.O and stock INTC.O has a correlation score of 0.9201
Stock INTC.O and stock AMZN.O has a correlation score of 0.9096
Stock INTC.O and stock .SPX has a correlation score of 0.9078
Stock INTC.O and stock SPY has a correlation score of 0.9076
```

In the following session, we are going to work with the pairs $(AAPL, MSFT)$ regarding to there correlation degree. First we plot the stock prices to show the correlation :

In [24]:

```
plt.plot(df['AAPL.O'],label= 'AAPL.O')
plt.plot(df['MSFT.O'],label= 'MSFT.O')
plt.ylabel('Price')
plt.xlabel('Date')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
seaborn.jointplot(df['AAPL.O'], df['MSFT.O'] ,color='b')
plt.show()
```





The correlation shows that the the two series can be written in the form : $Y = \alpha X + \beta$.

Now we use the statsmodels library to run a linear regression, a ordinary least square, between the two series to get α and β coefficients as follows :

In [25]:

```
df_tr_eikon = pd.read_csv(tr_eikon_path, index_col = 0, parse_dates = True).dropna()
symbols = ['AAPL.O','MSFT.O']

model = sm.OLS(df[symbols[0]], df[symbols[1]])
result = model.fit()
```

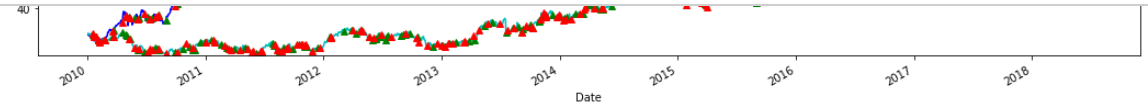
Now we implement the strategy :

In [26]:

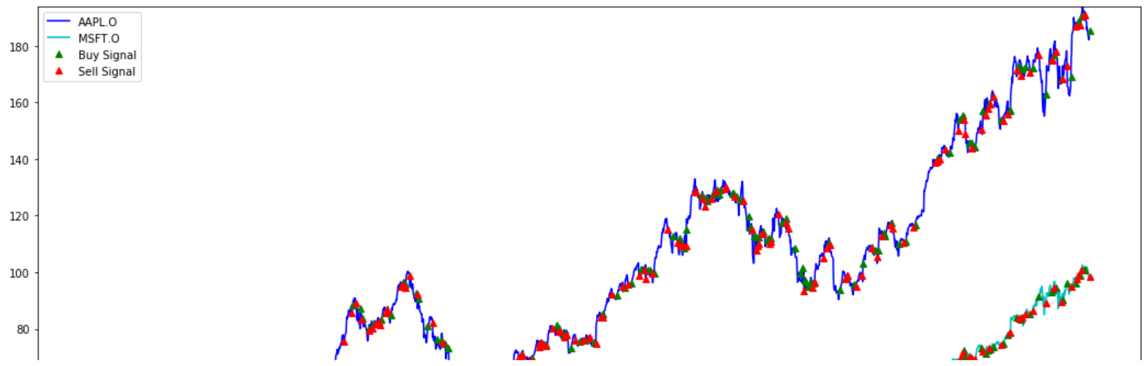
```
def pairsTradingStrategy(data, symbols, window, bollingerBand, exitScore = 0) :

    """ This function implements the strategy

    Parameters :
        - data : the dataframe with prices
        - window (int) : window for rolling the zScore
        - bollingerBand (float) : the bolliger band (characterizes the prices and volatilit
    Returns :
        - The profit of the strategy

    """

    # we first run the linear regression
    model = sm.OLS(data[symbols[0]], data[symbols[1]])
    result = model.fit()

    dataframe = pd.DataFrame({'stock1':data[symbols[0]],
                              'stock2':data[symbols[1]],
                              'spread': data[symbols[0]] - (data[symbols[0]] * result.param

    #computing the z score : normalization of the spread
    meanSpread = dataframe.spread.rolling(window=window).mean()
    stdSpread = dataframe.spread.rolling(window=window).std()
    zScore = (dataframe.spread - meanSpread) / stdSpread
    dataframe['zScore'] = zScore
    #we create a variable that's equal to 1 if both stocks are moving in the same direction
    # we initialize the values of our position on the stocks as being flat
    dataframe['orderStock1'] = 0
    dataframe['orderStock2'] = 0

    # we now implement the logic of the strategy ( 1 : buy, -1 : sell, 0 : flat)

    isEnteringUpperBand = ( dataframe['zScore'] >  bollingerBand) & (dataframe['zScore'].sh
    isNotEnteringUpperBand = ( dataframe['zScore'] >  bollingerBand) & (dataframe['zScore']

    isEnteringLowerBand = (dataframe['zScore'] < - bollingerBand) & (dataframe['zScore'].sh
    isNotEnteringLowerBand = (dataframe['zScore'] < - bollingerBand) & (dataframe['zScore']


    dataframe.loc[(isEnteringUpperBand), 'orderStock1'] = -1
    dataframe.loc[(isEnteringUpperBand), 'orderStock2'] = 1
    dataframe.loc[(isEnteringLowerBand), 'orderStock1'] = 1
    dataframe.loc[(isEnteringLowerBand), 'orderStock2'] = -1




    ####################

    #Fell free to decomment this part for plotting the ordersat time
    plt.figure(figsize=(18,9))
    S1 = dataframe.stock1
    S2 = dataframe.stock2
    S1.plot(color='b')
    S2.plot(color='c')
    buy = 0*S1.copy()
    sell = 0*S1.copy()
    buy[(isEnteringLowerBand)] = S1[(isEnteringLowerBand)]
    sell[(isEnteringLowerBand)] = S2[(isEnteringLowerBand)]
    buy[(isEnteringUpperBand)] = S2[(isEnteringUpperBand)]
    sell[(isEnteringUpperBand)] = S1[(isEnteringUpperBand)]
    buy.plot(color='g', linestyle='None', marker='^')
    sell.plot(color='r', linestyle='None', marker='^')
    x1,x2,y1,y2 = plt.axis()
    plt.axis((x1,x2,min(S1.min(),S2.min()),max(S1.max(),S2.max())))
    plt.legend([symbols[0],symbols[1], 'Buy Signal', 'Sell Signal'])
    plt.show()



    #############
    dataframe['stock1'] = np.log(dataframe['stock1'])
    dataframe['stock1'] = dataframe['stock1'].diff()
    dataframe['stock2'] = np.log(dataframe['stock2'])
    dataframe['stock2'] = dataframe['stock2'].diff()
    dataframe['profit'] = np.multiply(dataframe['stock1'], dataframe['orderStock1']) + np.m
    return dataframe, dataframe['profit'], dataframe['profit'].sum()
```

In [27]:

```
from mpl_toolkits.mplot3d import Axes3D

profits = []
windows = range(10,20)
bands = list(k/10 for k in range(-30, 30))
for window in windows:
    for band in bands:
        _, _, profit = pairsTradingStrategy(df, symbols, window, band/10)
        profits.append(profit)
        print ("Profit for window %d and band %.2f is : %.4f"%(window, band/10, profit))
```



```
Profit for window 10 and band -0.27 is : 3.5601
```

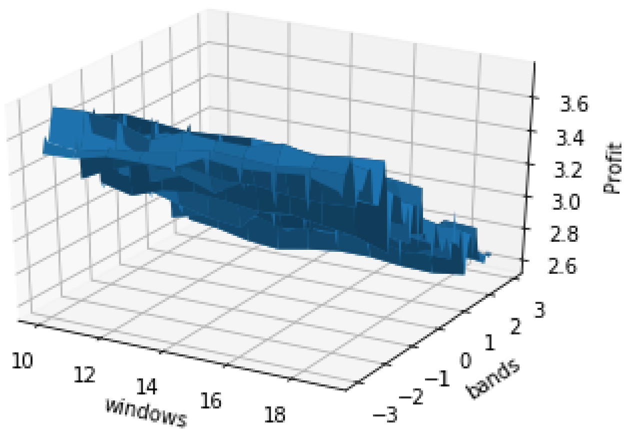


In [28]:

```
windows = np.expand_dims(np.array(windows), axis = -1)
X, Y = np.meshgrid(windows, bands)
Z = np.array(profits).reshape(X.shape)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z)


ax.set_xlabel('windows')
ax.set_ylabel('bands')
ax.set_zlabel('Profit')
```

Out[28]:

```
Text(0.5,0,'Profit')
```



The startegy above does not yield a negative profit, which shows that it is a market neutral risk strategy.

3. Comparing Strategies

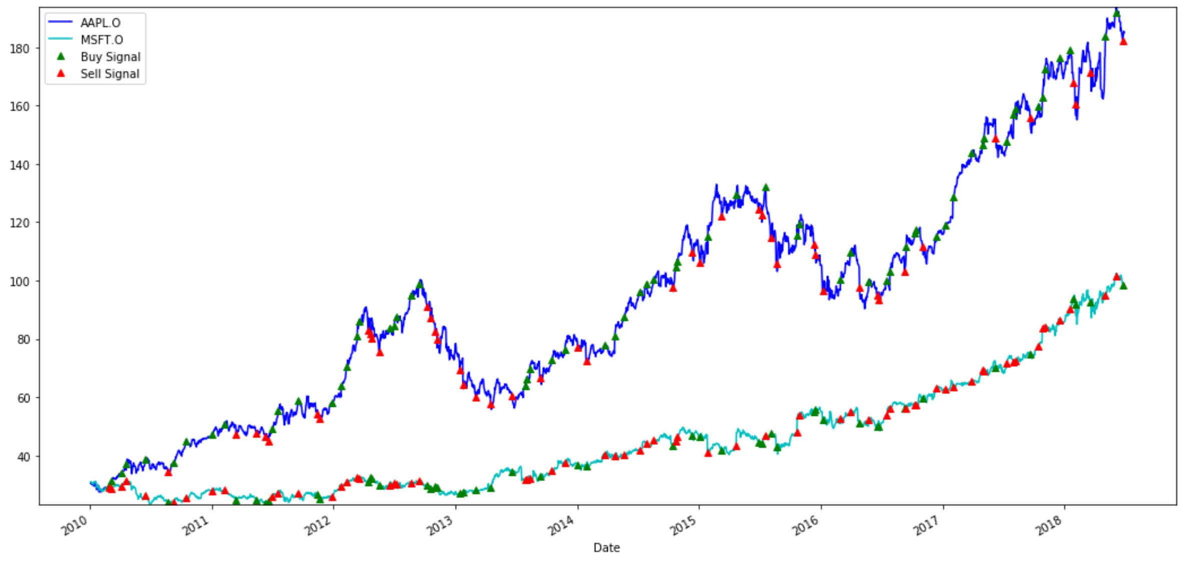
3.1 Profit over Risk Ratio :

In [29]:

```
def sh_ratio (returns):
    return  (returns.mean() / returns.std())
```

In [30]:

```
data = df_tr_eikon
fib_ratio = .618
symbols = ['AAPL.O','MSFT.O']
window = 20
bollingerBand = 2


fib_profit, fibonacci_profits, _, _, _ = fibonacciRetracementStrategy(fib_ratio, data , sym
sma_profit, sma_profits, _, _ = get_strategy_return(50, 160, dataframe, symbol = "AAPL.O",
_, pairTrading_profits, pairTrading_profit = pairsTradingStrategy(data, symbols, window, bo
fib_sharpe = sh_ratio(fibonacci_profits)
sma_sharpe = sh_ratio(sma_profits)
pairTrading_sharpe = sh_ratio(pairTrading_profits)

print ("Sharpe ratio for Fibonacci strategy is %.4f, Profit is %.4f"%(fib_sharpe, fib_profi
print ("Sharpe ratio for SMA strategy is %.4f, Profit is %.4f"%(sma_sharpe, sma_profit))
print ("Sharpe ratio for Pair Trading strategy is %.4f, Profit is %.4f"%(pairTrading_sharpe
```



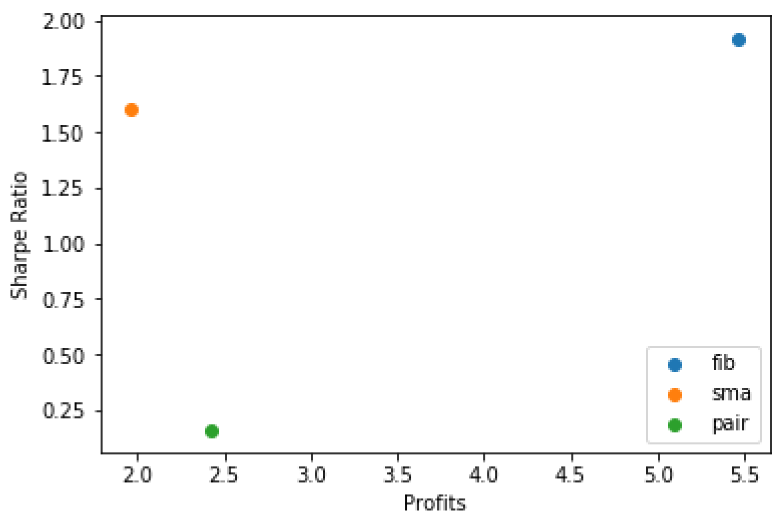
```
Sharpe ratio for Fibonacci strategy is 1.9182, Profit is 5.4682
Sharpe ratio for SMA strategy is 1.5992, Profit is 1.9599
Sharpe ratio for Pair Trading strategy is 0.1559, Profit is 2.4253
```

In [31]:

```
plt.scatter(fib_profit, fib_sharpe)
plt.scatter(sma_profit, sma_sharpe)
plt.scatter(pairTrading_profit, pairTrading_sharpe)
#plt.scatter(all_profits, all_ratios)
plt.xlabel("Profits")
plt.ylabel("Sharpe Ratio")
plt.legend(["fib", "sma", "pair"])
```

Out[31]:

```
<matplotlib.legend.Legend at 0x1d6c37bba20>
```



In [ ]:

```
```