Trading Algorithm. CORRELATION (Ejemplo base)

Pairs Trading

Pairs trading is a strategy that uses two stocks that are highly correlated. We can then use the difference in price between the two stocks as signal if one moves out of correlation with the other. It is an older strategy that is used classically as a guide to beginning algorithmic trading. There is a fantastic full guide and write up on Investopedia you can find here! **I highly recommend reading the article in full before continuing, it is entertaining and informative!**

Let's create our first basic trading algorithm! This is an exercise in using quantopian, **NOT** a realistic representation of what a good algorithm is! Never use something as simple as this in the real world! This is an extremely simplified version of Pairs Trading, we won't be considering factors such as cointegration!

Comodities en yahooFinances https://finance.yahoo.com/commodities

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import quandl
import yfinance as yf
import datetime as dt
```

In [2]:

```
#funcion que calcula un tipo de normalizacion (entiedo que hay muchos tipos, lo estudiaremos)
def zscore(stocks):
    return (stocks - stocks.mean()) / np.std(stocks)
```

In [3]:

```
# Funcion para preparar data series, buscaremos fechas en las que no tengamos los dos valores

def gapCleaning_beta (serie_A, serie_B,fechaST,fechaEND):
    #fecha   =dt.datetime(2008,11,18)
    #1- Longest serie?
    if (len(serie_A) > len(serie_B)):
```

```
        maxLengh = len(serie_A)
    else:
        maxLengh = len(serie_B)
    #return maxLengh

    #2- recorro los index y el que falte copio el valor del anterior
    for i in range (1000):
        #if (serie_A.index[i]  !=  serie_B.index[i]):

        try:
            print (serie_A.loc[fechaST]['Adj Close'])  #iloc serie_A.iloc[i]['Adj Close']
        except KeyError as ke:
            print("missing in index:", ke)

        fechaST= fechaST+ dt.timedelta(days=1)

def gapCleaning(serie_A, serie_B, fechaST, fechaEND):
    # Funcion que unifca timeSeries, agrupando fechas que tienen valor para ambos activos
    #print (serie_A.describe())   # explorar Datalore...

    mergedData=pd.merge(serie_A, serie_B, on='Date',how='inner')

    print (len(serie_A))
    print  (len(serie_B))
    print  (len(mergedData))
    #print (mergedData.describe())
    #return mergedData['Date']['Adj Close_x']
    return (mergedData[['Adj Close_x', 'Open_x']], mergedData[['Adj Close_y', 'Open_y']])
```

Instrumentos a correlar y rango de fechas

In [4]:

```
activo1 =  'ke=F'   #  'san' #'BZ=F'  #'ual'
activo2 = 'zr=F'  # 'bbva' #'GC=F' #'aal'

start   =dt.datetime(2000,11,18)
end   =dt.datetime(2022,8,1)
```

In [5]:

```
#serie1 = quandl.get('WIKI/UAL',start_date=start,end_date=end)
```

```
#serie2 = quandl.get('WIKI/AAL',start_date=start,end_date=end)

serie1 = yf.download(activo1, start,end)
serie2 = yf.download(activo2, start,end)
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

In [6]:

```
serie1.info()
serie2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5450 entries, 2000-11-17 to 2022-07-29
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       5450 non-null   float64
 1   High       5450 non-null   float64
 2   Low        5450 non-null   float64
 3   Close      5450 non-null   float64
 4   Adj Close  5450 non-null   float64
 5   Volume     5450 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 298.0 KB
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 4957 entries, 2000-11-17 to 2022-07-29
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       4957 non-null   float64
 1   High       4957 non-null   float64
 2   Low        4957 non-null   float64
 3   Close      4957 non-null   float64
 4   Adj Close  4957 non-null   float64
 5   Volume     4957 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 271.1 KB
```

In [7]:

```
serie1, serie2 = gapCleaning(serie1,serie2,start,end)
serie1.rename(columns={"Adj Close_x":"Adj Close"}, inplace=True)
serie2.rename(columns={"Adj Close_y":"Adj Close"}, inplace=True)
```

```
5450
4957
4949
```

In [8]:

```python
len(serie1)
```

Out[8]:

```
4949
```

In [9]:

```python
serie2.head()
```

Out[9]:

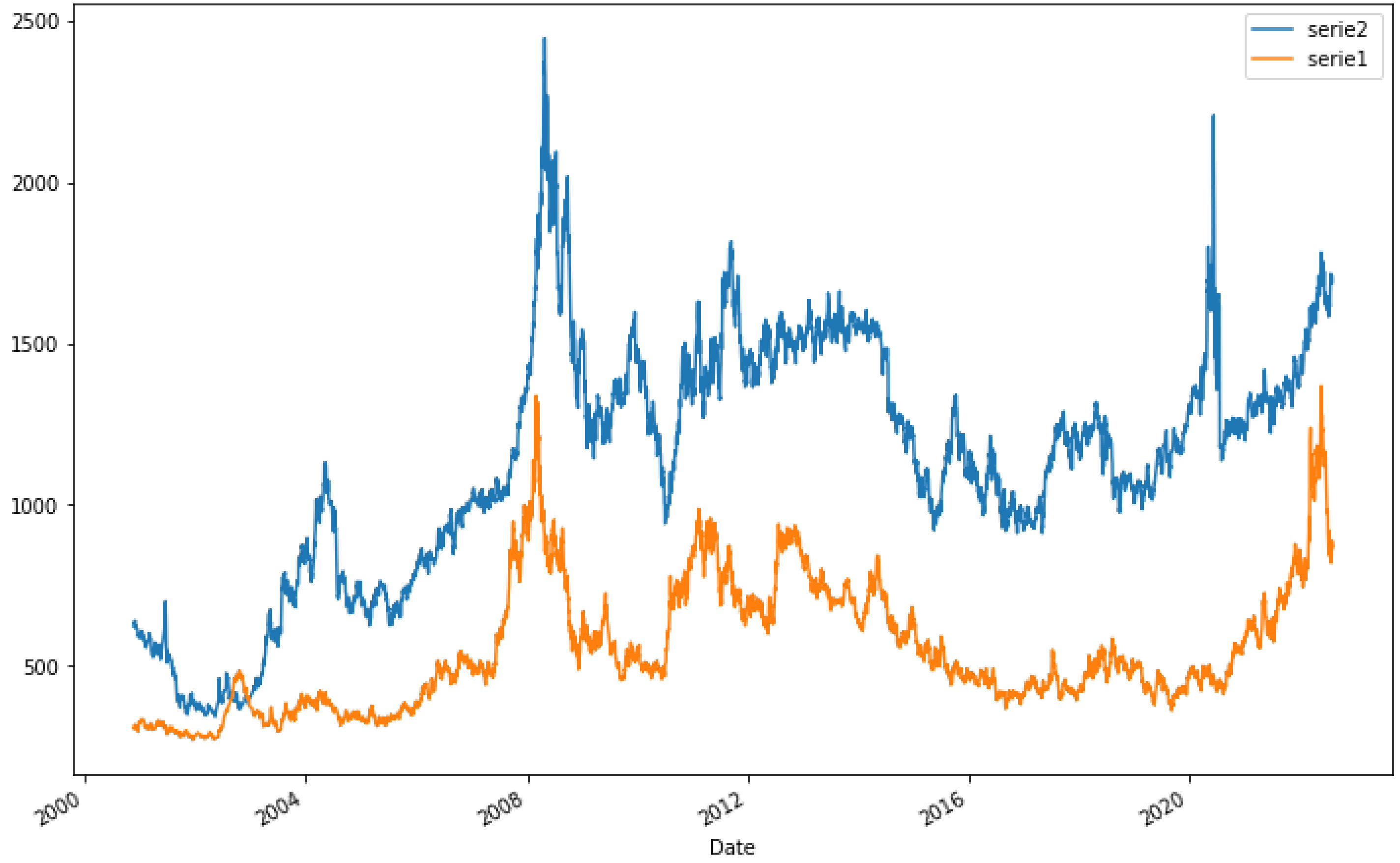
	Adj Close	Open_y
Date		
2000-11-17	624.0	624.0
2000-11-20	625.0	625.0
2000-11-21	636.0	636.0
2000-11-22	634.0	634.0
2000-11-24	638.0	638.0

In [10]:

```python
serie2['Adj Close'].plot(label='serie2 ',figsize=(12,8))
serie1['Adj Close'].plot(label='serie1 ')
plt.legend()
```

Out[10]:

```
<matplotlib.legend.Legend at 0x2544c435a60>
```



Spread and Correlation

Coeficiente de correlación producto-momento de Pearson En las estadísticas, el coeficiente de correlación producto-momento de Pearson es una medida de la correlación lineal entre dos variables X e Y, dando un valor entre +1 y -1 inclusive, donde 1 es correlación positiva total, 0 es ninguna correlación y -1 Es la correlación negativa total.

https://es.wikipedia.org/wiki/Correlaci%C3%B3n

In [11]:

```
serie2['Adj Close']
```

Out[11]:

```
Date
2000-11-17     624.0
```

```
2000-11-20     625.0
2000-11-21     636.0
2000-11-22     634.0
2000-11-24     638.0
                ...
2022-07-25    1693.0
2022-07-26    1709.5
2022-07-27    1708.5
2022-07-28    1702.0
2022-07-29    1688.5
Name: Adj Close, Length: 4949, dtype: float64
```

In [12]:

```
serie2.loc[:,'Adj Close']
```

Out[12]:

```
Date
2000-11-17     624.0
2000-11-20     625.0
2000-11-21     636.0
2000-11-22     634.0
2000-11-24     638.0
                ...
2022-07-25    1693.0
2022-07-26    1709.5
2022-07-27    1708.5
2022-07-28    1702.0
2022-07-29    1688.5
Name: Adj Close, Length: 4949, dtype: float64
```

In [13]:

```
serie2_Norm= zscore(serie2['Adj Close'])
```

In [14]:

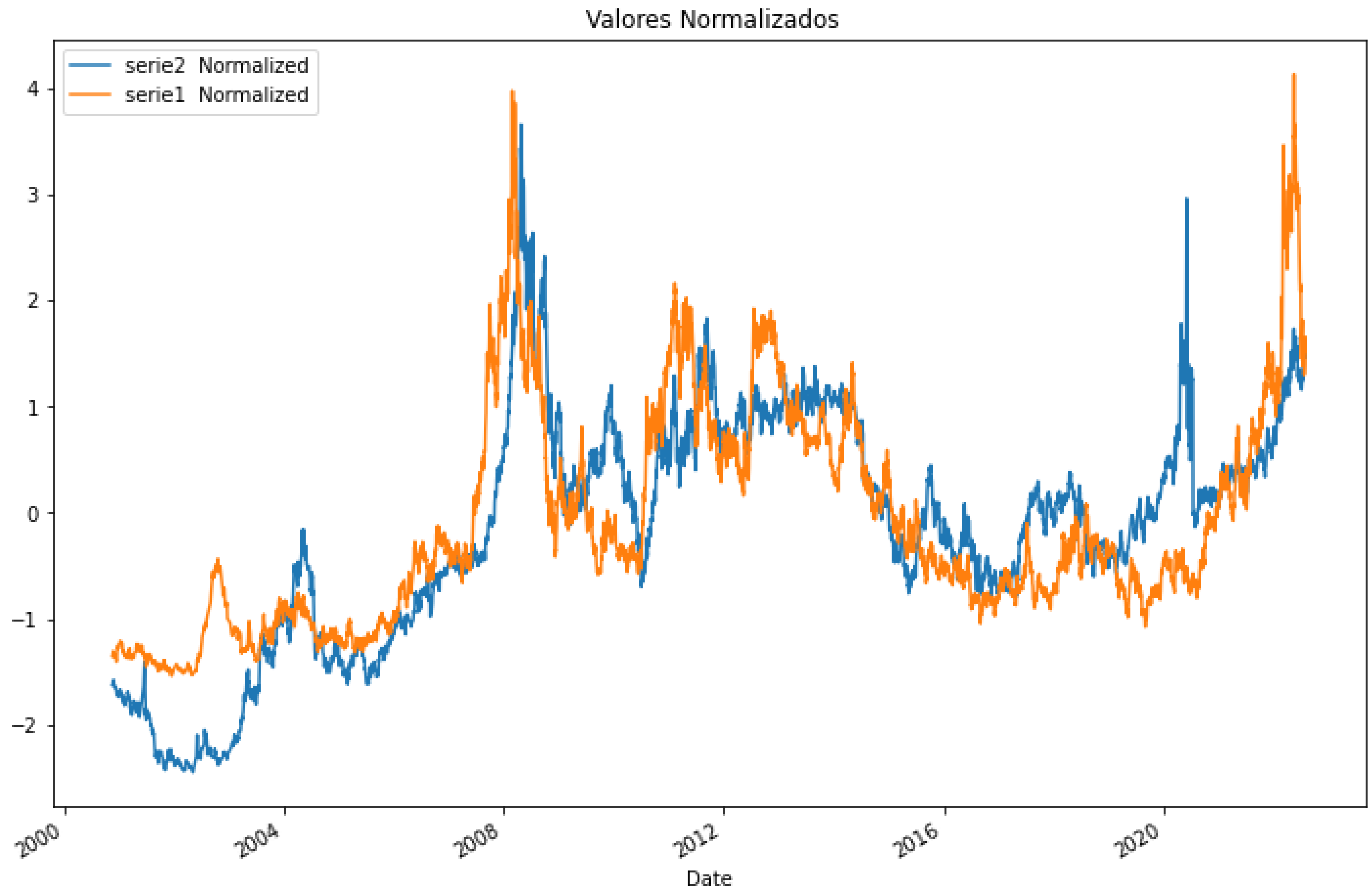
```
serie1_Norm= zscore(serie1['Adj Close'])
```

In [15]:

```
serie2_Norm.plot(label='serie2  Normalized',figsize=(12,8))
serie1_Norm.plot(label='serie1  Normalized')
plt.legend()
plt.title(' Valores Normalizados')
```

Out[15]:

```
Text(0.5, 1.0, ' Valores Normalizados')
```



In [16]:
```
np.corrcoef(serie2['Adj Close'],serie1['Adj Close'])
```

Out[16]:
```
array([[1.        , 0.77505071],
       [0.77505071, 1.        ]])
```

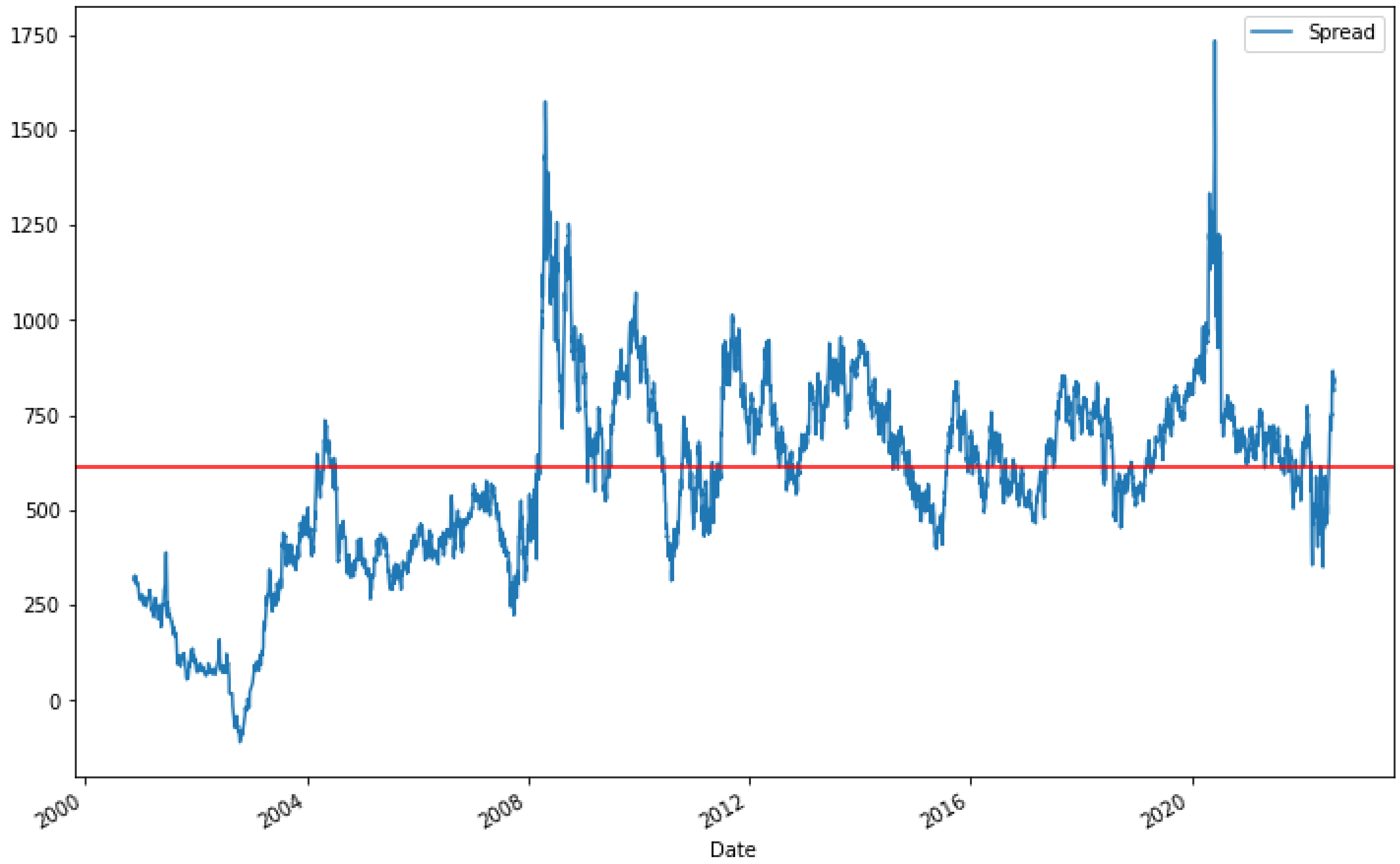
El Coeficiente nos calcula la relacion entre las series de datos y nos ofrece una matriz de resultados de correlaciones. En este caso columnas AA - UA y filas AA -UA La diagonal nos ofrece un dato =1, al compara la serie con ella misma, las otras celdas muestran las realciones entre las distintas series.

In [17]:
```
array = np.corrcoef(serie2_Norm,serie1_Norm)
#array[0][0]
```

In [18]:

```python
spread = serie2['Adj Close'] - serie1['Adj Close']
spread.plot(label='Spread',figsize=(12,8))
plt.axhline(spread.mean(),c='r')
plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x2544df4cd60>



In [19]:
```python
serie2.head()
```

Out[19]:

	Adj Close	Open_y
Date		
2000-11-17	624.0	624.0

	Adj Close	Open_y
Date		
2000-11-20	625.0	625.0
2000-11-21	636.0	636.0
2000-11-22	634.0	634.0
2000-11-24	638.0	638.0

In [20]:

```
#serie2.loc[:,'Adj Close']
serie2.loc['2019-01-01':'2019-01-10','Adj Close']
```

Out[20]:

```
Date
2019-01-02    1008.5
2019-01-03    1017.0
2019-01-04    1022.5
2019-01-07    1040.5
2019-01-08    1049.0
2019-01-10    1059.5
Name: Adj Close, dtype: float64
```

In [21]:

```
serie1.loc['2019-01-01':'2019-01-10','Adj Close']
```

Out[21]:

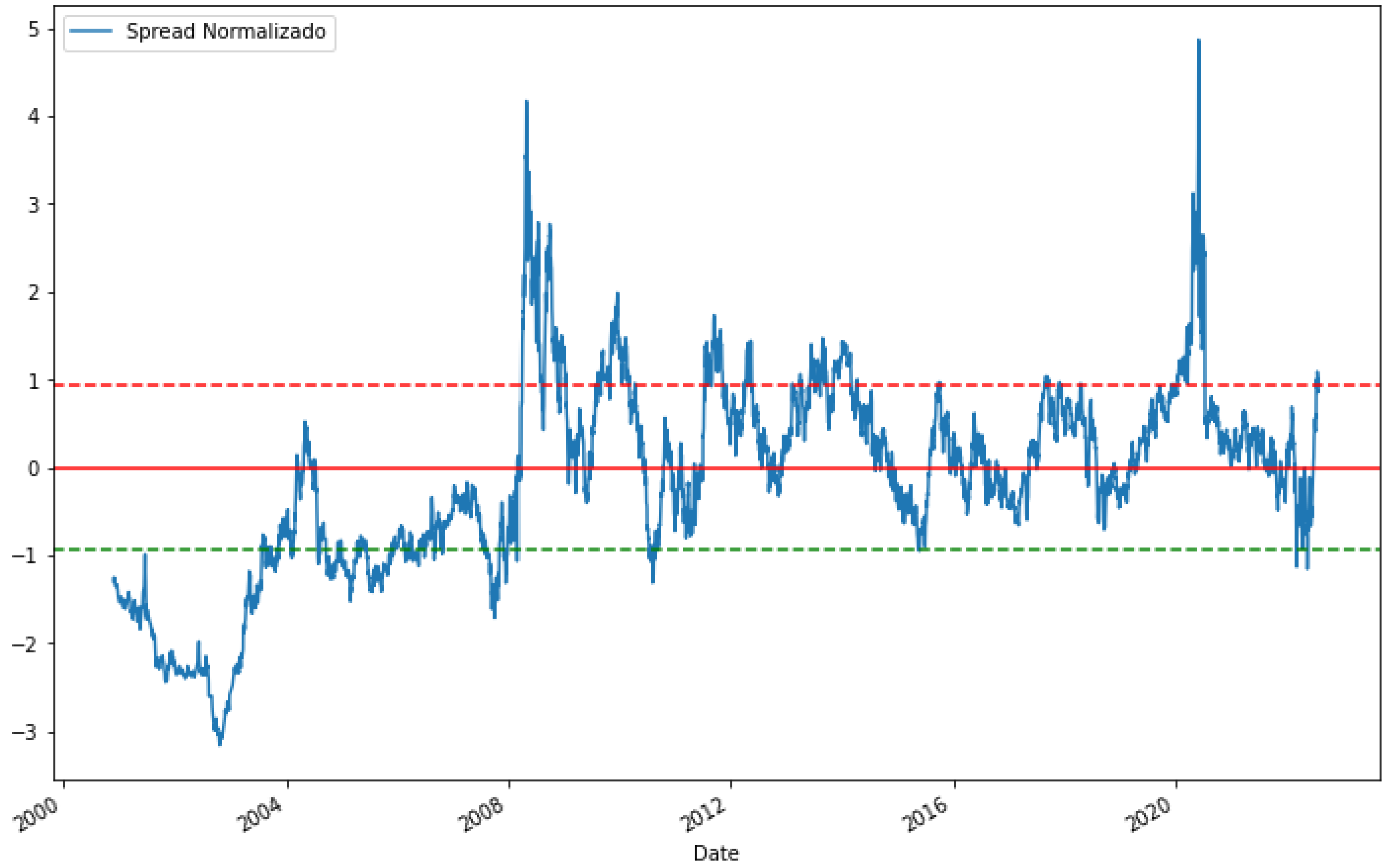
```
Date
2019-01-02    492.50
2019-01-03    503.50
2019-01-04    506.00
2019-01-07    503.00
2019-01-08    505.00
2019-01-10    498.75
Name: Adj Close, dtype: float64
```

In [22]:

```
lag = start
spread_fast = serie2.loc[lag:end,['Adj Close']] - serie1.loc[lag:end,['Adj Close']]  #Sofia_Grades = Report_Card.iloc[6:12,2:]
spread_fast_norm= zscore(spread_fast['Adj Close'])

spread_fast_norm.plot(label='Spread Normalizado',figsize=(12,8))
plt.axhline(spread_fast_norm.mean(),c='r')
plt.axhline(0.93, c='r', ls='--')
```

```
plt.axhline(-0.93, c='g', ls='--')
plt.legend()
```

Out[22]: <matplotlib.legend.Legend at 0x2544e470700>

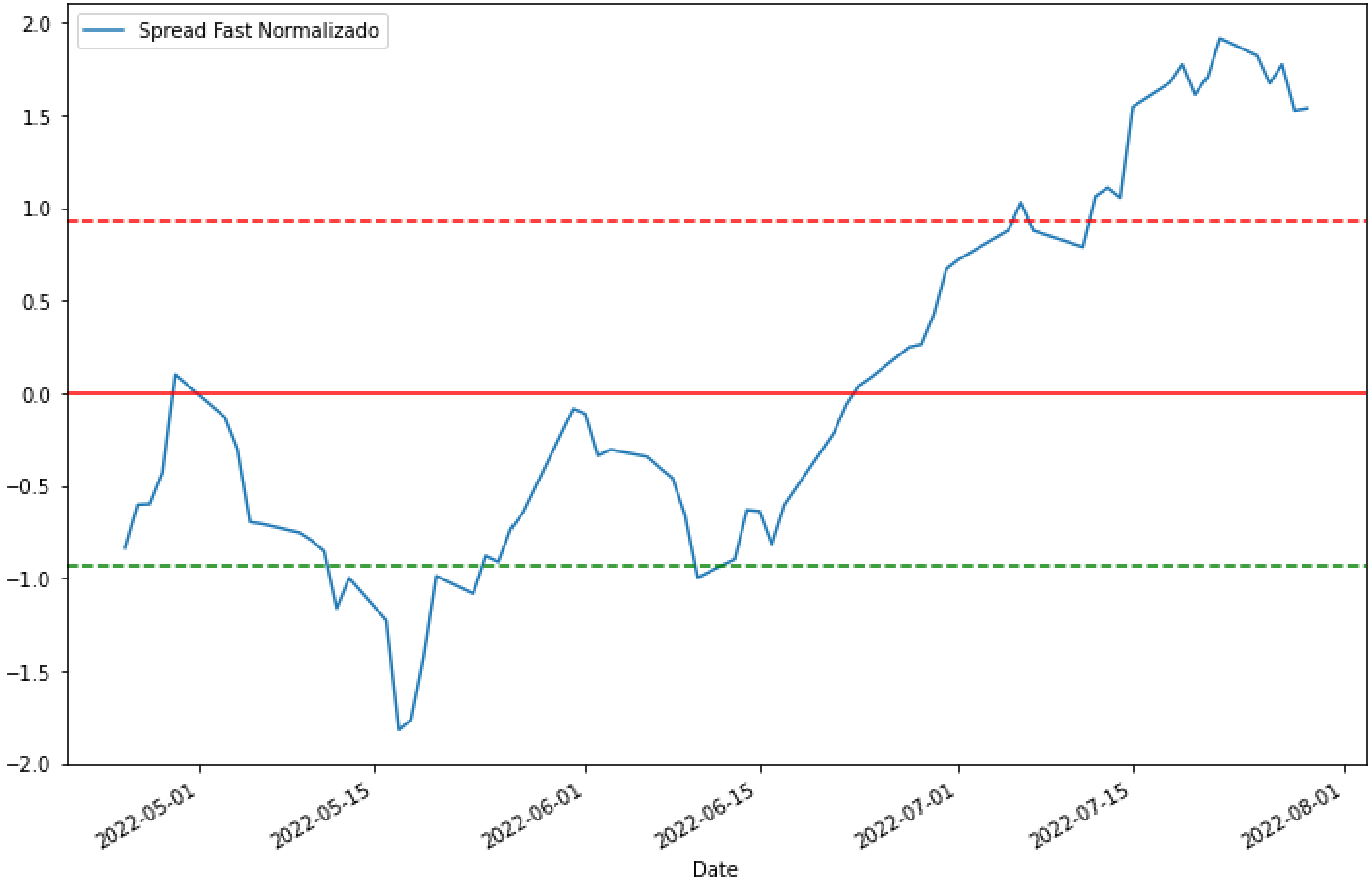


In [23]:

```
#spread_fast = serie2.loc['2017-06-01':'2019-01-01',['Adj Close','Open_y']] - serie1.loc['2017-06-01':'2019-01-01',['Adj Close','O

lag = end - dt.timedelta(days=100)
spread_fast = serie2.loc[lag:end,['Adj Close']] - serie1.loc[lag:end,['Adj Close']]  #Sofia_Grades = Report_Card.iloc[6:12,2:]
spread_fast_norm= zscore(spread_fast['Adj Close'])

spread_fast_norm.plot(label='Spread Fast Normalizado',figsize=(12,8))
plt.axhline(spread_fast_norm.mean(),c='r')
plt.axhline(0.93, c='r', ls='--')
```

```
plt.axhline(-0.93, c='g', ls='--')
plt.legend()
```

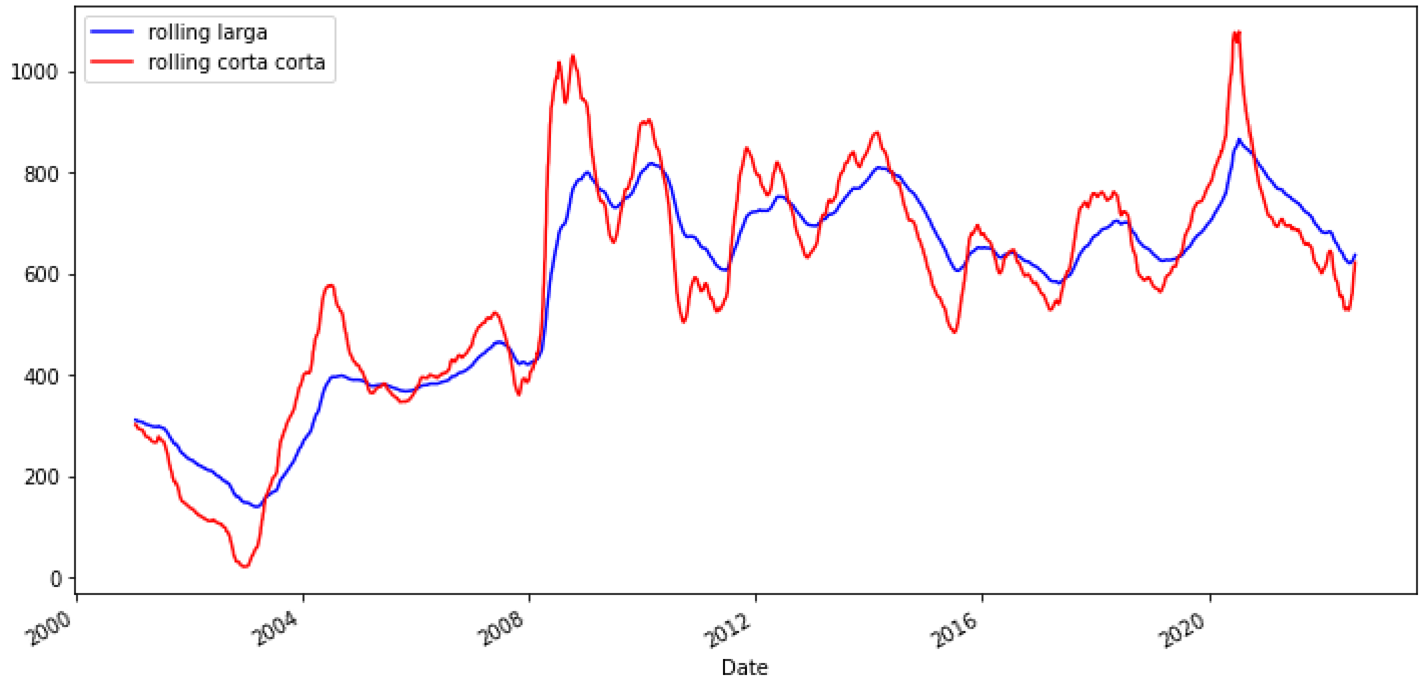
Out[23]: <matplotlib.legend.Legend at 0x2544e217eb0>



In [24]:

```
lag = end - dt.timedelta(days=500)
emea=pd.Series.ewm(spread,200, min_periods=30,adjust=False).mean()
emea_c=pd.Series.ewm(spread,100, min_periods=30,adjust=False).mean()
emea_cc=pd.Series.ewm(spread,50, min_periods=30,adjust=False).mean()
emea.loc[start:end] .plot(label='rolling larga',figsize=(12,6),c='b')
#emea_c.loc[start:end].plot(label='rolling corta',c='g')
emea_cc.loc[start:end].plot(label='rolling corta corta',c='r')
plt.legend()
```

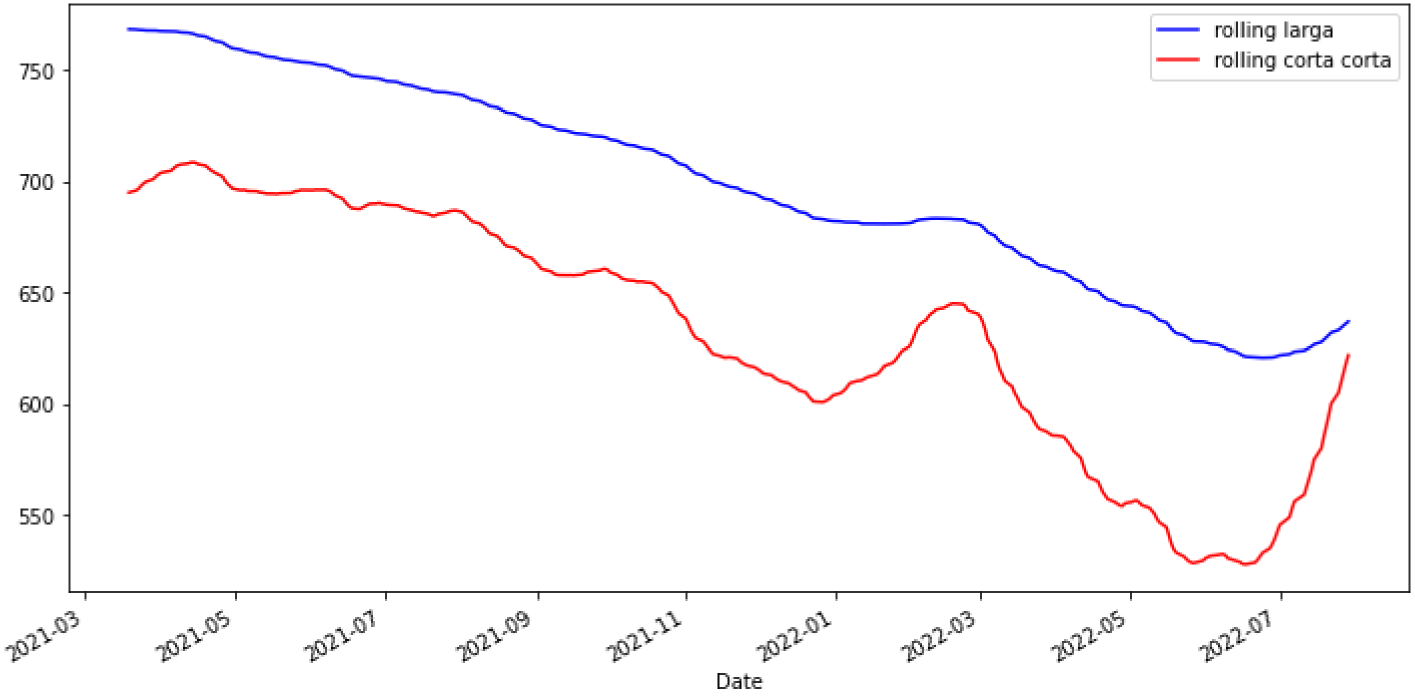
Out[24]: <matplotlib.legend.Legend at 0x2544e470250>



In [25]:

```
emea=pd.Series.ewm(spread,200, min_periods=30,adjust=False).mean()
emea_c=pd.Series.ewm(spread,100, min_periods=30,adjust=False).mean()
emea_cc=pd.Series.ewm(spread,50, min_periods=30,adjust=False).mean()
emea.loc[lag:end] .plot(label='rolling larga',figsize=(12,6),c='b')
#emea_c.loc[lag:end].plot(label='rolling corta',c='g')
emea_cc.loc[lag:end].plot(label='rolling corta corta',c='r')
plt.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x2544ea297c0>



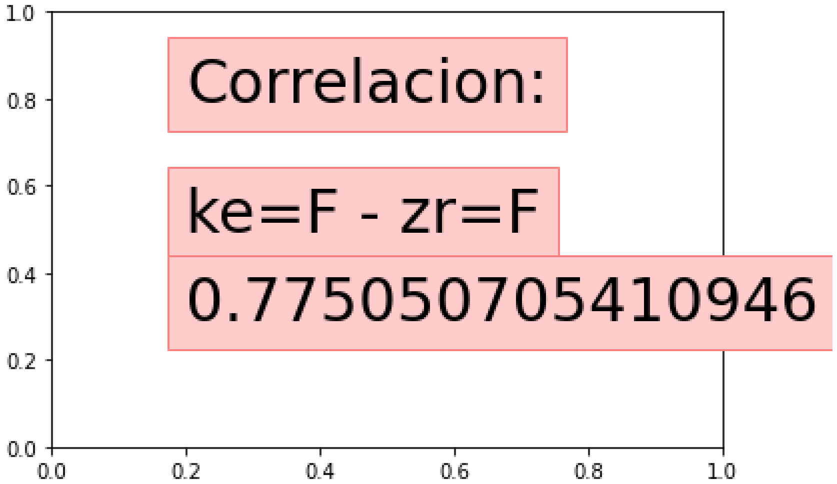
In [26]:

```
texto= array[0][1] # +activo1+activo2
plt.text(0.2, 0.6, activo1 + ' - '+ activo2 , size=30, rotation=0.,
         ha="left", va="top",
         bbox=dict(boxstyle="square",
                   ec=(1., 0.5, 0.5),
                   fc=(1., 0.8, 0.8),
                   )
         )
plt.text(0.2, 0.4, texto , size=30, rotation=0.,
         ha="left", va="top",
         bbox=dict(boxstyle="square",
                   ec=(1., 0.5, 0.5),
                   fc=(1., 0.8, 0.8),
                   )
         )
plt.text(0.2, 0.9, "Correlacion:" , size=30, rotation=0.,
         ha="left", va="top",
         bbox=dict(boxstyle="square",
                   ec=(1., 0.5, 0.5),
                   fc=(1., 0.8, 0.8),
```

```
                    )
          )
```

Out[26]:

```
Text(0.2, 0.9, 'Correlacion:')
```

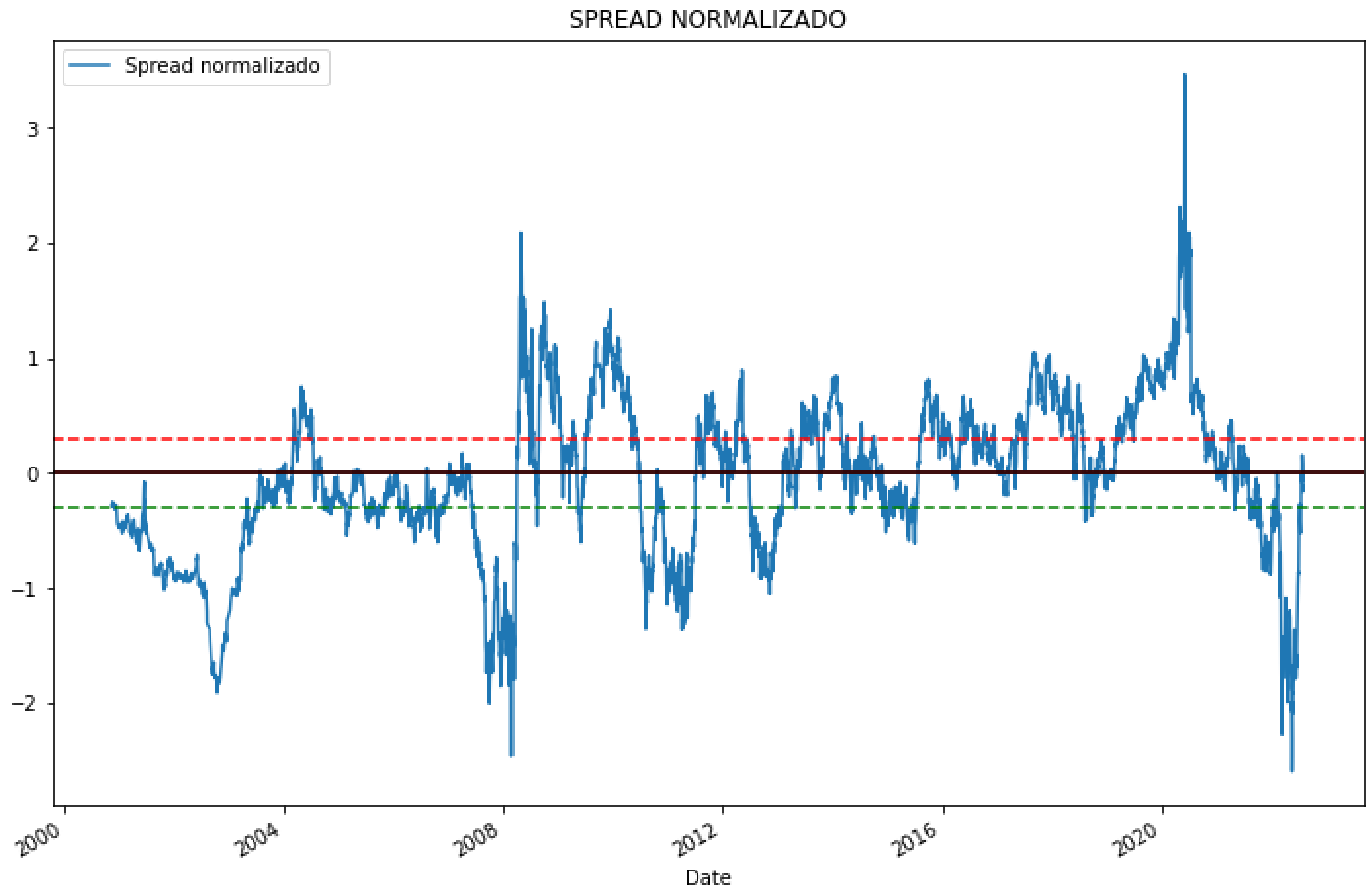


In [27]:

```
spread_nor = serie2_Norm - serie1_Norm
spread_nor.plot(label='Spread normalizado',figsize=(12,8))
plt.axhline(spread_nor.mean(),c='r')
plt.axhline(zscore(spread).mean(), color='black')

plt.axhline(0.3, c='r', ls='--')
plt.axhline(-0.3, c='g', ls='--')
plt.legend()
plt.title('SPREAD NORMALIZADO')
```

Out[27]:

```
Text(0.5, 1.0, 'SPREAD NORMALIZADO')
```



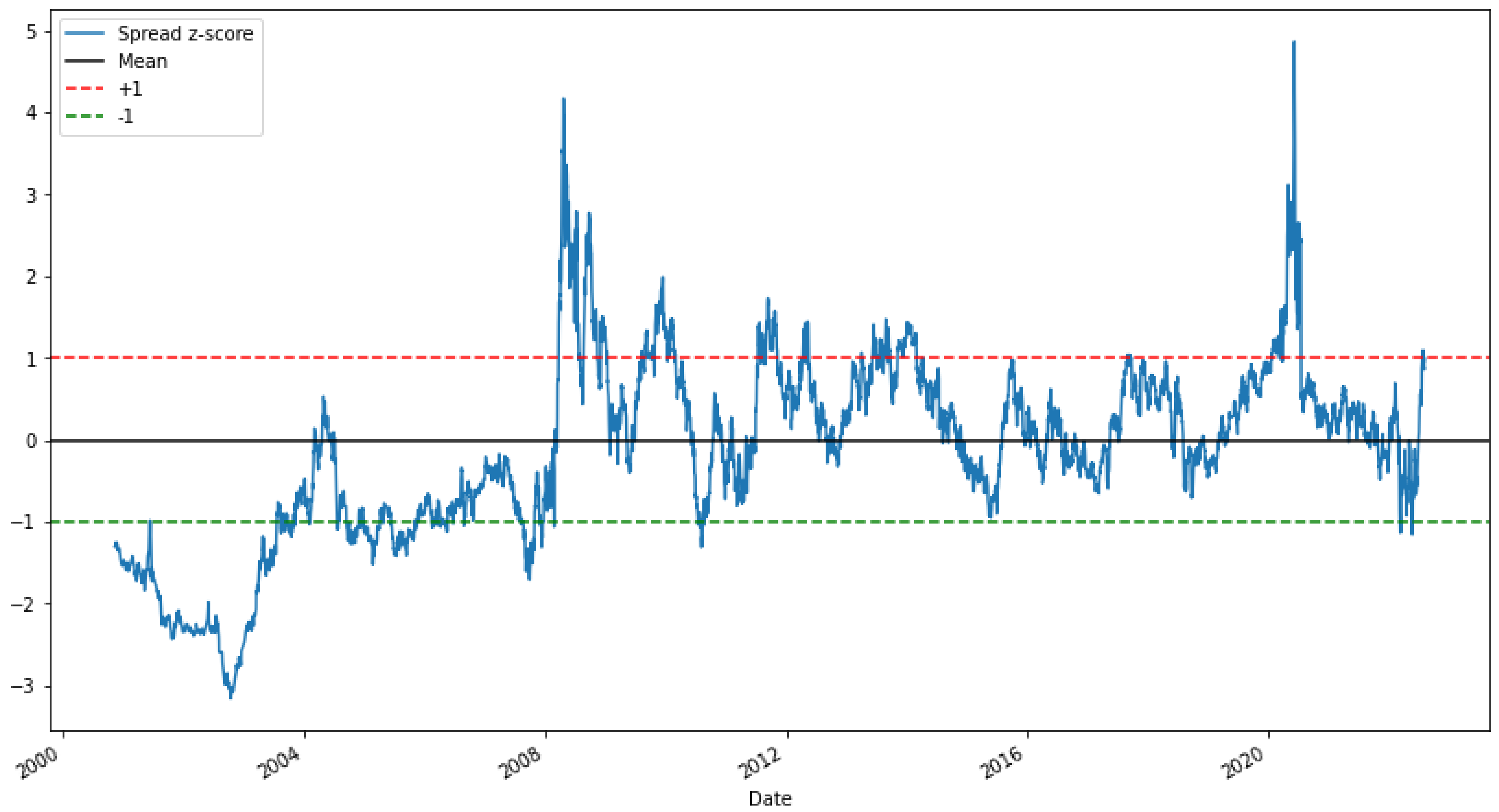
Por Spread entendemos la diferencia entre los valores...

Normalizing with a z-score

def zscore(stocks): return (stocks - stocks.mean()) / np.std(stocks)

In [28]:

```
zscore(spread).plot(figsize=(14,8))

plt.axhline(zscore(spread).mean(), color='black')

plt.axhline(1.0, c='r', ls='--')
plt.axhline(-1.0, c='g', ls='--')
plt.legend(['Spread z-score', 'Mean', '+1', '-1']);
```



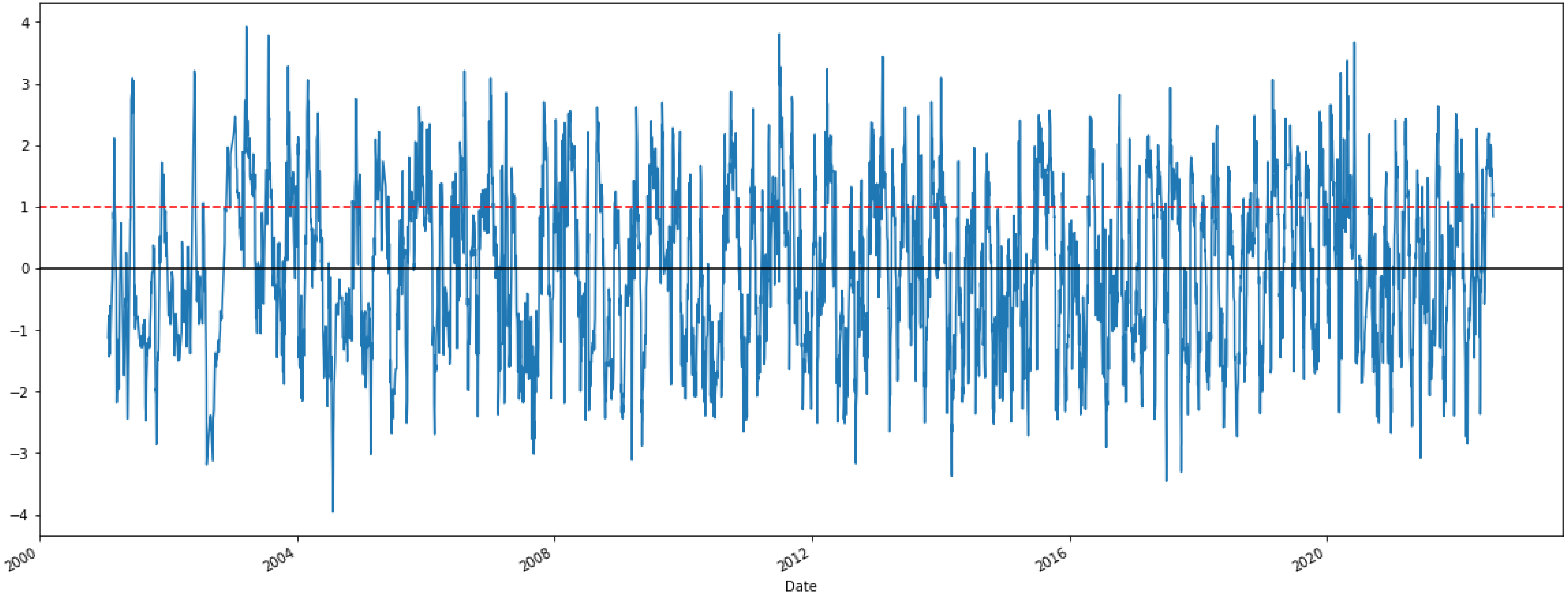
Rolling Z-Score

Our spread is currently serie2-serie1. Let's decide how to calculate this on a rolling basis for our use in Quantopian

In [29]:

```
#1 day moving average of the price spread
spread_mavg1 = spread.rolling(1).mean()

# 30 day moving average of the price spread
spread_mavg30 = spread.rolling(30).mean()

# Take a rolling 30 day standard deviation
std_30 = spread.rolling(30).std()
```

```python
# Compute the z score for each day
zscore_30_1 = (spread_mavg1 - spread_mavg30)/std_30



zscore_30_1.plot(figsize=(20,8),label='Rolling 30 day Z score')
plt.axhline(0, color='black')
plt.axhline(1.0, color='red', linestyle='--');
```



Implementation of Strategy

WARNING: YOU SHOULD NOT ACTUALLY TRADE WITH THIS!

In [30]:

```python
import numpy as np

def initialize(context):
    """
    Called once at the start of the algorithm.
```

```
    """

    # Every day we check the pair status
    schedule_function(check_pairs, date_rules.every_day(), time_rules.market_close(minutes=60))

    # Our Two Airlines
    context.aa = sid(45971) #aal
    context.ual = sid(28051) #ual

    # Flags to tell us if we're currently in a trade
    context.long_on_spread = False
    context.shorting_spread = False


def check_pairs(context, data):

    # For convenience
    aa = context.aa
    ual = context.ual

    # Get pricing history
    prices = data.history([aa, ual], "price", 30, '1d')


    # Need to use .iloc[-1:] to get dataframe instead of series
    short_prices = prices.iloc[-1:]

    # Get the long 30 day mavg
    mavg_30 = np.mean(prices[aa] - prices[ual])

    # Get the std of the 30 day long window
    std_30 = np.std(prices[aa] - prices[ual])

    # Get the shorter span 1 day mavg
    mavg_1 = np.mean(short_prices[aa] - short_prices[ual])

    # Compute z-score
    if std_30 > 0:
        zscore = (mavg_1 - mavg_30)/std_30

        # Our two entry cases
        if zscore > 0.5 and not context.shorting_spread:
            # spread = aa - ual
            order_target_percent(aa, -0.5) # short top
```

```
            order_target_percent(ual, 0.5) # long bottom
            context.shorting_spread = True
            context.long_on_spread = False

        elif zscore < -0.5 and not context.long_on_spread:
            # spread = aa - ual
            order_target_percent(aa, 0.5) # long top
            order_target_percent(ual, -0.5) # short bottom
            context.shorting_spread = False
            context.long_on_spread = True

        # Our exit case
        elif abs(zscore) < 0.1:
            order_target_percent(aa, 0)
            order_target_percent(ual, 0)
            context.shorting_spread = False
            context.long_on_spread = False

        record('zscore', zscore)
```

In [ ]:

In [ ]: