QUANTSTART

Basics of Statistical Mean Reversion Testing - Part II

Basics of Statistical Mean Reversion Testing - Part II

Updated for Python 3.9, January 2023

I'd like to thank Dr. Tom Starke for providing the inspiration for this article series. The code below is a modification of that which used to be found on his website leinenbock.com, which later became drtomstarke.com.

A while back we began discussing statistical mean reversion testing. In that article we looked at a couple of techniques that helped us determine whether a time series was mean reverting or not. In particular we looked at the Augmented Dickey-Fuller Test and the Hurst Exponent. In this article we will consider another test for mean reversion, namely the **Cointegrated Augmented Dickey Fuller** (CADF) test.

Firsty, it should be noted that it is actually very difficult to find a directly tradable asset that possesses mean-reverting behaviour. For instance, equities broadly behave like GBMs and hence render the mean-reverting trade strategies relatively useless. However, there is nothing stopping us from creating a *portfolio* of price series that is stationary. Hence we can apply mean-reverting trading strategies to the portfolio.

The simplest form of mean-reverting trade strategies is the classic "pairs trade", which usually involves a dollar-neutral long-short pair of equities. The theory goes that two companies in the same sector are likely to be exposed to similar market factors, which affect their businesses. Occasionally their relative stock prices will diverge due to certain events, but will revert to the long-running mean.

Let's consider two energy sector equities Exxon Mobil Corp given by the ticker XOM and United States Oil Fund given by the ticker USO. Both are exposed to similar market conditions and thus will likely have a stationary pairs relationship. We are now going to create some plots, using Pandas and Matplotlib to demonstrate the cointegrating nature of XOM and USO. Full code for creating these figures is given below along with all the details necessary to follow along with the analysis. The first plot (Figure 1) displays their respective price histories for the period Jan 1st 2019 to Jan 1st 2020.

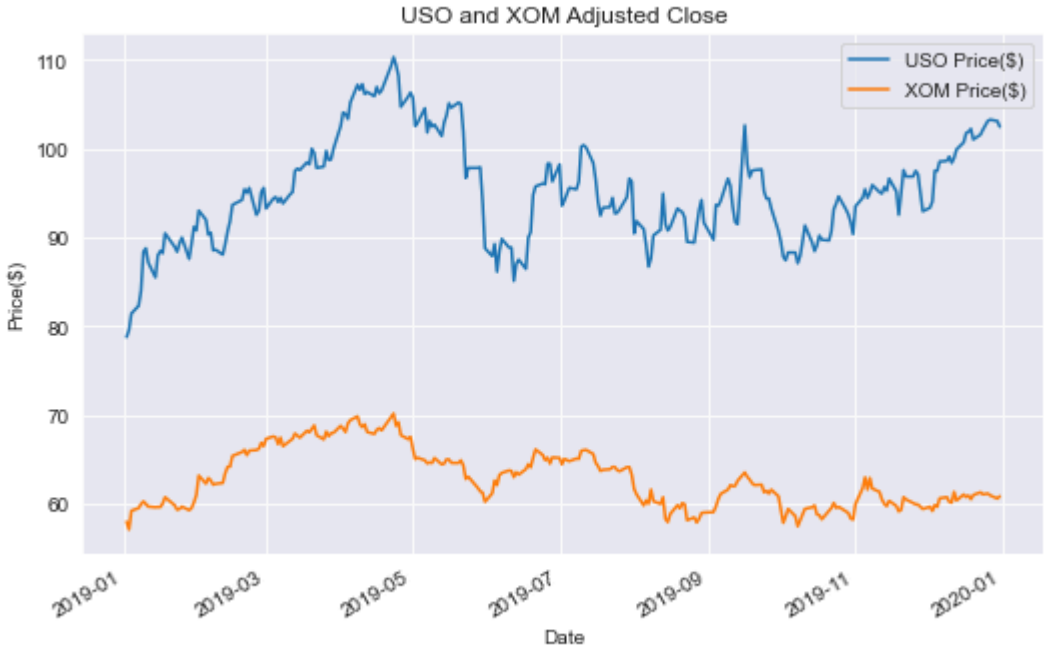

Fig 1 - Time series plots of XOM and USO

If we create a scatter plot of their prices, we see that the relationship is broadly linear (see Figure 2) for this period.

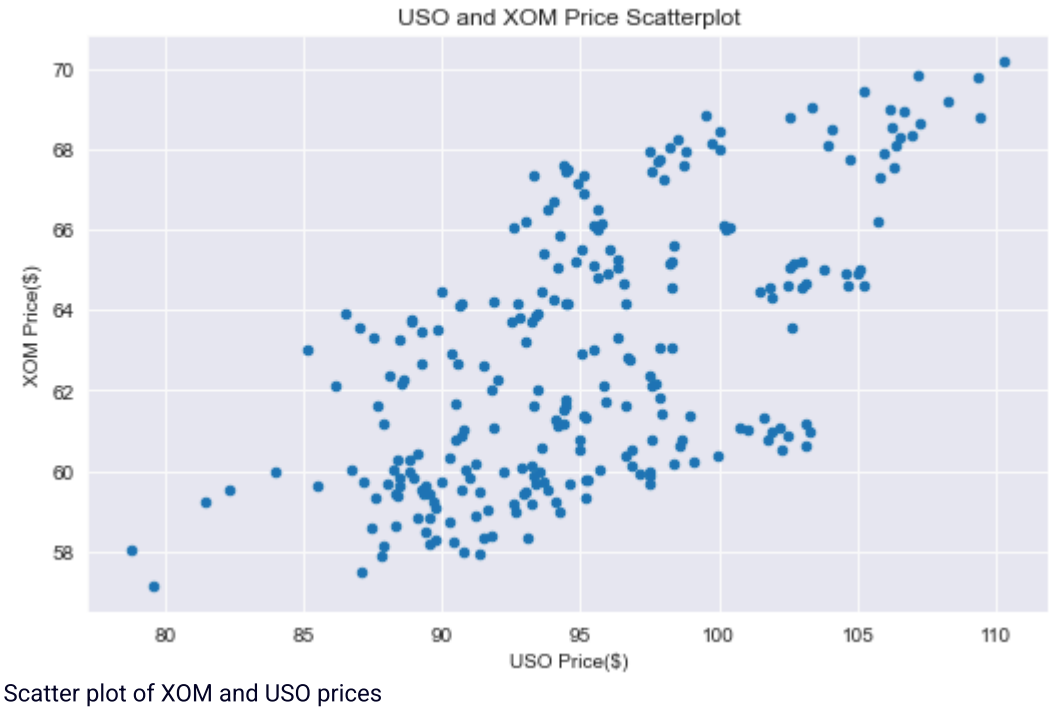

Fig 2 - Scatter plot of XOM and USO prices

The pairs trade essentially works by using a linear model for a relationship between the two stock prices:

$$\hat{y}_t = \beta x_t + \epsilon_t$$

Where $\hat{y}_t$ is the price of USO stock and x_t is the price of XOM stock, both on day t.

The residuals are the difference between the predicted values of the y_t and the observed values, $r_t = y_t - \hat{y}_t$. Given that the regression line has the equation $\hat{y}_t = \beta x_t + \epsilon_t$ we can calculate the residual of an observation as follows:

$$r_t = y_t - \hat{y}_t = y_t - (\beta x_t + \epsilon_t)$$

If we plot the residuals (for a particular value of β that we will determine below) we create a new time series that, at first glance, does not look particularly stationary. This is given in Figure 3:

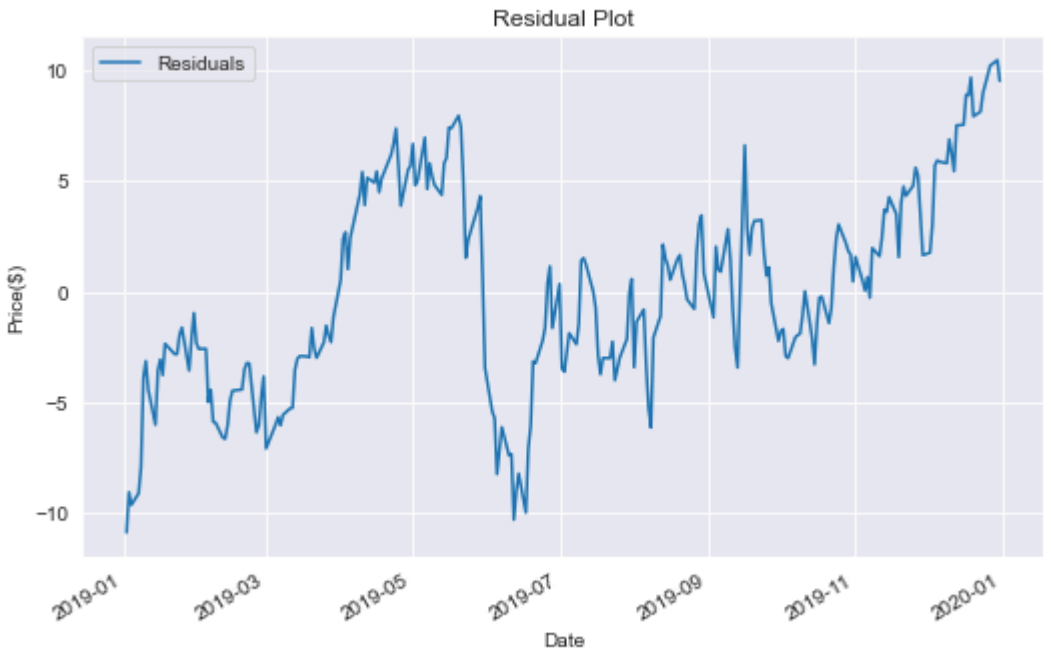

Fig 3 - Residual plot of USO and XOM linear combination

Cointegrated Augmented Dickey-Fuller Test

In order to statistically confirm whether this series is mean-reverting we could use one of the tests that we considered in the previous article, namely the Augmented Dickey-Fuller Test or the Hurst Exponent. However, neither of these tests will actually help us determine β, the hedging ratio needed to form the linear combination, they will only tell us whether, for a particular β, the linear combination is stationary.

This is where the Cointegrated Augmented Dickey-Fuller (CADF) test comes in. It determines the optimal hedge ratio by performing a linear regression against the two time series and then tests for stationarity under the linear combination.

Python Implementation

We will now use Python libraries to test for a cointegrating relationship between USO and XOM for the period of Jan 1st 2019 to Jan 1st 2020. We will use Python v3.8, Pandas v1.3, Matplotlib v3.4 and Statsmodels v0.12 to carry out the ADF test, as above.

In order to follow along you will need to obtain OHLCV data in csv format for both XOM and USO for the period 1st Jan 2019 to 1st Jan 2020. To run the code you will need to update the path to contain the location of your csv files.

The first task is to create a new file, `cadf.py`, and import the necessary libraries. Following this we will use the Pandas function `read_csv()` to create two DataFrames for the OHLCV data for both XOM and USO. We ensure that our index column is of type Datetime by using the `parse_dates` keyword. We then concatenate the two DataFrames on their index and extract the Adjusted Close price for both USO and XOM. These will be our pairs equities. The procedure is wrapped up in a `__main__` function which we will update throughout.

```
# cadf.py

import matplotlib.pyplot as plt
import os
import pandas as pd
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts


def create_price_dataframe(path):
    """
    Read pricing data csv download for USO and XOM
    OHLCV data from 01/01/2019-01/01/2020 into DataFrames.

    Parameters
    ----------
    path : `str`
        Directory location of CSV files containing USO and XOM data.

    Returns
    -------
    price_df : `pd.DataFrame`
        A DataFrame containing XOM and USO Adjusted Close data from
        01/01/2019-01/01/2020. Index is a Datetime object.

    """
    uso = pd.read_csv(
        os.path.join(csv_path, "USO.csv"),
        index_col=0,
        parse_dates=True
    )
    xom = pd.read_csv(
        os.path.join(csv_path, "XOM.csv"),
        index_col=0,
        parse_dates=True
    )

    # Select columns to add to new DataFrame
    price_data = [uso["Adj Close"], xom["Adj Close"]]
    # Create headers for the columns
    headers = ["USO Price($)", "XOM Price($)"]
    # Concatenate xom and uso DataFrames using the index column
    price_df = pd.concat(price_data, axis=1, keys=headers)
    return price_df


if __name__ == "__main__":
    csv_path = "PATH/TO/YOUR/CSV"

    price_dataframe = create_price_dataframe(csv_path)
```

Our final DataFrame `price_dataframe` looks as follows. We can use `price_df.head()` to see the first few rows:

```
            USO Price($)    XOM Price($)
Date
2019-01-02  78.800003   58.018692
2019-01-03  79.599998   57.127888
2019-01-04  81.440002   59.234173
2019-01-07  82.320000   59.542213
2019-01-08  84.000000   59.975113
```

The second function, `plot_price_series`, takes the `price_df` DataFrame as input. The function simply plots the two price series on the same chart. This allows us to visually inspect whether any cointegration may be likely.

We make use of the Pandas `price_df.plot()` function. As we have used the Pandas keyword argument `parse_dates=True` to set up our DataFrames the plotting function is able to correctly display the dates for the price series. We use Matplotlib's `set_ylabel()` to correctly label the Y axis and `plt.show()` to display the figure. Finally we add a call to the function into our `__main__`:

```
# cadf.py

def plot_price_series(price_df):
    """
    Plot the Adjusted Close price series for XOM and USO.

    Parameters
    ----------
    price_df : `pd.DataFrame`
        A DataFrame containing XOM and USO Adjusted Close data from
        01/01/2019-01/01/2020. Index is a Datetime object.

    Returns
    -------
    None
    """
    fig = price_df.plot(title="USO and XOM Daily Prices")
    fig.set_ylabel("Price($)")
    plt.show()


if __name__ == "__main__":
    csv_path = "PATH/TO/YOUR/CSV"

    price_dataframe = create_price_dataframe(csv_path)
    # NEW
    plot_price_series(price_dataframe)
```

The third function, `plot_scatter_series`, plots a scatter plot of the two prices. This allows us to visually inspect whether a linear relationship exists between the two series and thus whether it is a good candidate for the OLS procedure and subsequent ADF test:

```
# cadf.py

def plot_scatter_series(price_df):
    """
    Plot the Scatter plot of the XOM and USO price series.

    Parameters
    ----------
    price_df : `pd.DataFrame`
        A DataFrame containing XOM and USO Adjusted Close data from
        01/01/2019-01/01/2020. Index is a Datetime object.

    Returns
    -------
    None
    """
    price_df.plot.scatter(x=0, y=1, title="USO and XOM Price Scatterplot")
    plt.show()


if __name__ == "__main__":
    csv_path = "PATH/TO/YOUR/CSV"

    price_dataframe = create_price_dataframe(csv_path)
    plot_price_series(price_dataframe)
    # NEW
    plot_scatter_series(price_dataframe)
```

The fourth function, `create_residuals`, calculates the residuals by calling the Statsmodels OLS function on the XOM and USO series. This allows us to calculate the β hedge ratio. The hedge ratio is then used to create a "Residuals" column via the formation of the linear combination of both XOM and USO.

```
# cadf.py

def create_residuals(price_df):
    """
    Calculate the OLS and create the beta hedge ratio and residuals for the two
    equites XOM and USO.

    Parameters
    ----------
    price_df : `pd.DataFrame`
        A DataFrame containing XOM and USO Adjusted Close data from
        01/01/2019-01/01/2020. Index is a Datetime object.

    Returns
    -------
    price_df : `pd.DataFrame`
        Updated DataFrame with column values for beta hedge ratio (beta_hr) and
        residuals (Residuals).
    """
    # Create OLS model
    Y = price_df['USO Price($)']
    x = price_df['XOM Price($)']
    x = sm.add_constant(x)
    model = sm.OLS(Y, x)
    res = model.fit()

    # Beta hedge ratio (coefficent from OLS)
    beta_hr = res.params[1]
    print(f'Beta Hedge Ratio: {beta_hr}')

    # Residuals
    price_df["Residuals"] = res.resid
    return price_df


if __name__ == "__main__":
    csv_path = "PATH/TO/YOUR/CSV"

    price_dataframe = create_price_dataframe(csv_path)
    plot_price_series(price_dataframe)
    plot_scatter_series(price_dataframe)
    # NEW
    residuals_dataframe = create_residuals(price_dataframe)
```

Finally the ADF test is carried out on the calculated residuals and the result is printed. We then plot the residuals using the `plot_residuals` function. This is designed to plot the residual values from the fitted linear model of the two price series. This function requires that the pandas DataFrame has a "Residuals" column, representing the residual prices:

```
# cadf.py

def create_cadf(price_df):
    """
    Calculate the Cointegrated Augmented Dickey Fuller test on the residuals.

    Parameters
    ----------
    price_df : `pd.DataFrame`
        Updated DataFrame with column values for beta hedge ratio (beta_hr) and
        residuals (Residuals).

    Returns
    -------
    cadf : `tuple`
        Results of ADF test on residuals including the test statistic,
        pvalue and critical values.
    """
    cadf = ts.adfuller(price_df["Residuals"])
    print(f'CADF:{cadf}')
    return cadf


def plot_residuals(price_df):
    """
    Plot the residuals.

    Parameters
    ----------
    price_df : `pd.DataFrame`
        Updated DataFrame with column values for beta hedge ratio (beta_hr) and
        residuals (Residuals).

    Returns
    -------
    None
    """
    plt.figure()
    price_df.plot(y="Residuals", title="Residual Plot", figsize=(8.6, 5.3))
    plt.ylabel("Price($)")
    plt.show()


if __name__ == "__main__":
    csv_path = "PATH/TO/YOUR/CSV"

    price_dataframe = create_price_dataframe(csv_path)
    plot_price_series(price_dataframe)
    plot_scatter_series(price_dataframe)
    residuals_dataframe = create_residuals(price_dataframe)
    # NEW
    cadf_dataframe = create_cadf(residuals_dataframe)
    plot_residuals(residuals_dataframe)
```

The output of the code (along with the Matplotlib plots) is as follows:

```
CADF:(-2.891342330777582,
0.046364069139156444, 0, 251,
{'1%': -3.4566744514553016, '5%': -2.8731248767783426, '10%': -2.5729436702592023}, 878.843778326628)
```

It can be seen that the calculated test statistic of -2.891 is more negative than the 5% critical value of -2.873, which means that we can reject the null hypothesis that there isn't a cointegrating relationship at the 5% level. Hence we can conclude, with a reasonable degree of certainty, that USO and XOM possess a cointegrating relationship, at least for the time period sample considered.

Why Statistical Testing?

Fundamentally, as far as algorithmic trading is concerned, the statistical tests outlined above are only as useful as the profits they generate when applied to trading strategies. Thus, surely it makes sense to simply evaluate performance at the strategy level, as opposed to the price/time series level? Why go to the trouble of calculating all of the above metrics when we can simply use trade level analysis, risk/reward measures and drawdown evaluations?

Firstly, any implemented trading strategy based on a time series statistical measure will have a far larger sample to work with. This is simply because when calculating these statistical tests, we are making use of each *bar* of information, rather than each *trade*. There will be far less round-trip trades than bars and hence the statistical significance of any trade-level metrics will be far smaller.

Secondly, any strategy we implement will depend upon certain parameters, such as look-back periods for rolling measures or z-score measures for entering/exiting a trade in a mean-reversion setting. Hence strategy level metrics are only appropriate *for these parameters*, while the statistical tests are valid for the underlying time series sample.

In practice we want to calculate both sets of statistics. Python, via the statsmodels and pandas libraries, make this extremely straightforward. The additional effort is actually rather minimal!

Related Articles

- Backtesting An Intraday Mean Reversion Pairs Strategy Between SPY And IWM
- Basics of Statistical Mean Reversion Testing