Chapter 8 Statistical Arbitrage with Hypothesis Testing

Statistical arbitrage is a market-neutral trading strategy leveraging statistical methods to identify and exploit significant relationships between financial assets. Through hypothesis testing, it discerns pricing discrepancies within correlated asset pairs due to temporary market inefficiencies. By purchasing underpriced and selling overpriced assets, the strategy ensures profit as the market corrects these inefficiencies, regardless of overall market movements.

Statistical arbitrage

Statistical arbitrage refers to the use of statistical methods to identify statistically significant relationships underlying multiple financial assets and generate trading signals. There are two parts involved in this process: statistical analysis and arbitrage. In this context, statistical analysis mostly refers to hypothesis testing, which is a suite of statistical procedures that allows us to determine if a specific relationship among multiple financial instruments based on the observed data is statistically significant. On the other hand, arbitrage means making sure-win profits.

At its core, this strategy relies on mean-reversion, which assumes that financial instruments that have deviated far from their historical relationship will eventually converge again. For instance, consider two highly correlated stocks, A and B. If, due to some short-term market factors, the price of A increases disproportionately compared to B, a statistical arbitrage strategy might involve short-selling A (which is now overpriced) and buying B (which is underpriced). As the prices of A and B revert to their historical correlation, the arbitrageur would close both positions - buy A to cover the short sell and sell B to realize the gain. The net profit comes from the convergence of prices. Therefore, statistical arbitrage is essentially a market-neutral strategy, generating profits by taking advantage of temporary market inefficiencies.

Note that statistical arbitrage strategies should expect a relatively stable long-term equilibrium relationship between the two underlying assets for the strategy to work. They also operate on relatively small profit margins, necessitating high volumes of trades to generate substantial returns.

Delving deeper, the first step in the statistical arbitrage process is to identify pairs of trading instruments that exhibit a high degree of co-movement. This can be achieved

through statistical procedures such as correlation analysis or cointegration tests. For instance, consider stocks A and B, which typically move in sync with each other. Although perfect correlation is rare in financial markets, we can leverage historical price data to find stocks that are highly correlated, often within the same industry or sector.

However, this co-movement doesn't always mean equal price changes. Short-term fluctuations driven by various factors like market sentiment, sudden news announcements, or unforeseen events like a pandemic, can cause a temporary divergence in the price relationship. In the given example, if stock A increases by 10% and stock B only by 5%, it suggests a temporary mispricing where B is underpriced relative to A.

This brings us to the second step, which involves capitalizing on this mispricing through trading actions such as pairs trading. In the case of A and B, an investor could execute a long position on the underpriced stock B, expecting its price to increase and converge with the price of A.

It's important to note that statistical arbitrage relies heavily on the premise that these pricing inefficiencies are temporary and that the price relationship will revert to its historical norm. Therefore, this strategy necessitates diligent monitoring and a robust risk management system to ensure timely entries and exits.

Figure 8-1 illustrates one way of performing statistical arbitrage. We assume a perfect correlation between stocks A and B, where the same percentage change is observed for periods 0, 1, and 2. However, stock A increased by 10% in period 3, while stock B only increased by 5%. Based on the principle of statistical arbitrage, we could long stock B, which is considered to be underpriced, or short stock A, which is considered overpriced. We could also do both at the same time.

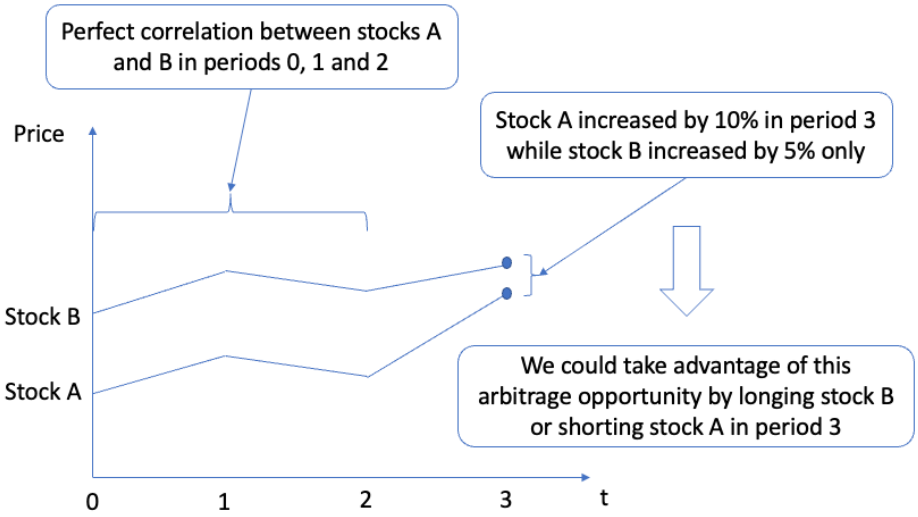

Figure 8-1 Illustrating the concept of statistical arbitrage. After identifying a perfect correlation between stocks A and B using statistical techniques, as indicated by the prices in periods 0, 1, and 2, we would take advantage of market mispricing by longing stock B (which is underpriced) and/or shorting stock A (which is overpriced).

Pairs trading

Pairs trading is a market-neutral strategy that leverages statistical analysis to generate potential profits regardless of the overall market direction. The "pair" in pairs trading refers to simultaneously taking two positions: going long on one asset and short on another, with the key requirement being that these assets are highly correlated. The trading signal stems from the spread or price difference between these two assets.

An unusually large spread, in comparison to historical data, suggests a temporary divergence, and the anticipation is that this divergence will eventually correct itself, reverting to its mean or average value over time. Traders can capitalize on this mean-reverting behavior, initiating trades when the spread is abnormally wide, and closing them once the spread narrows and returns to its typical range.

The determination of what constitutes an "abnormal" or "normal" spread is crucial and forms the core parameters of the pairs trading strategy. This typically involves extensive back-testing, where historical price data is analyzed to identify consistent patterns in price divergence and convergence, which then informs the thresholds for trade entry and exit points. Pairs trading, while robust in its market-neutral stance, requires a keen understanding of the long-term equilibrium relationship between the paired assets and careful management of potential risks if the expected price convergence does not materialize.

In the strategy of pairs trading, asset selection is grounded in a statistical procedure called hypothesis testing, specifically, the cointegration test. This process uses historical price data to identify pairs of financial instruments that exhibit a high level of correlation. When two assets are highly correlated, they tend to move in a synchronized manner. This means that any price change in one asset is typically mirrored proportionally by the other, resulting in relatively stable spreads that do not deviate significantly from their historical average. However, there can be moments when this spread deviates markedly from its historical norm, suggesting temporary mispricing of the assets. This divergence indicates that the assets' prices have drifted apart more than their usual correlation would predict.

Such deviations create a unique profit opportunity in pairs trading. Traders can capitalize on these large spreads by betting on their future contraction. Specifically, the strategy would be to go long on the underpriced asset and short on the overpriced one, with the anticipation that the spread will revert back to its historical average as the asset prices correct themselves. This reversion provides the opportunity to close both positions at a profit.

Figure 8-2 provides the overall workflow of implementing a pairs trading strategy. At first, we analyze a group of financial assets (such as stocks) and identify a pair that passes the cointegration test. This is a statistical test that determines if a group of assets is cointegrated, meaning their combination generates a stationary time series, despite each individual time series not exhibiting such stationarity. In other words, the historical differences, or spreads, of the two cointegrated assets form a stationary time series. We can thus monitor the current spread and check if it exceeds a reasonable range of historical

spreads. Exceeding the normal range indicates a trading signal to enter two positions: long the underpriced asset and short the overpriced asset. We would then hold these positions until the current spread shrinks back to the normal range, upon which point we would exit the positions and lock in a profit before it shrinks even further (which results in a loss).

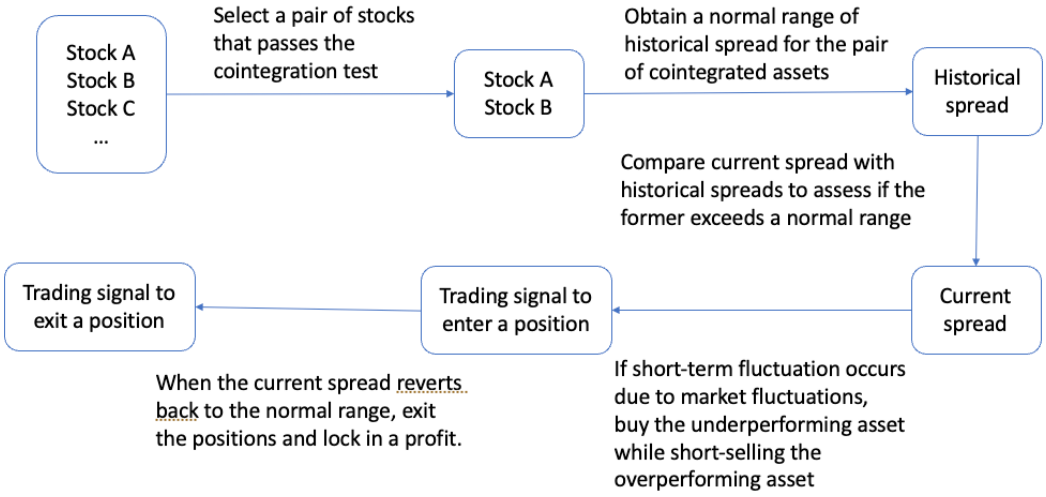

Figure 8-2 Overall workflow of implementing the pairs trading strategy.

Cointegration

Cointegration, a concept pivotal to hypothesis testing, posits two potential scenarios: the null hypothesis, which states that two or more non-stationary time series are not cointegrated, and the alternative hypothesis, which claims the opposite, i.e., these time series are cointegrated if their linear combination generates a stationary time series (more on this later).

Let's demystify some of the jargon here. A time series refers to a sequence of data points indexed (or listed or graphed) in time order, with each data point assigned a specific timestamp. This dataset can be analyzed through several summary statistics or statistical properties. These can include metrics like mean and variance computed over a certain time frame or window.

Moving this window across different periods, a stationary time series exhibits constancy in its mean and variance on average. This means that no matter when you observe it, its basic properties do not change. On the other hand, a nonstationary time series demonstrates a trend or a drift, signifying a changing mean and variance across varying time periods. These time series are dynamic, with their basic properties shifting over time, often due to factors like trends and seasonality.

Hence, the process of cointegration examines whether there is a long-term equilibrium relationship between non-stationary time series despite short-term fluctuations. Such long-term equilibrium manifests as a stationary time series as a linear combination of the two non-stationary time series.

Many traditional statistical methods, including Ordinary Least Squares (OLS) regression, are based on the assumption that the variables under analysis — which are also time series data points — exhibit stationarity. This implies that their fundamental statistical characteristics remain consistent over time. However, when dealing with non-stationary variables, this stationarity assumption gets violated. As a result, different techniques are needed to perform the modeling. One common strategy is to difference the non-stationary variable (deriving a new time series by taking the difference in the observed values of two consecutive time points) to eliminate any observable trend or drift.

A non-stationary time series might possess a unit root, which signifies a root of 1 in its autoregressive (AR) polynomial. To put it differently, the value in the next time period is strongly impacted by the present period value. This dependency reflects a form of serial correlation, where values from previous periods exert influence on subsequent ones, thereby potentially leading to non-stationary behavior.

The unit root test, therefore, is a method to examine whether a time series is non-stationary and possesses a unit root. Identifying and addressing the presence of a unit root is a critical step in the process of time series modeling, especially when the aim is to understand long-term trends and forecasts.

In essence, a cointegration test examines the assumption that, although individual time series may each have a unit root and hence be non-stationary, a linear combination of these time series might result in a stationary series. This forms the alternative hypothesis for the test.

To be precise, the alternative hypothesis states that the aggregate time series, derived from a linear combination of individual time series, achieves stationarity. Should this be the case, it would imply a persistent long-term relationship among these time series variables. Such long-term relationships will get obscured by temporary fluctuations in the market from time to time, due to factors such as mispricing. Hence, the cointegration test aids in revealing these hidden long-term relationships among time series variables.

When assets are determined to be cointegrated — meaning that the alternative hypothesis is upheld — they are fed into the trading signal generation phase of the pairs trading strategy. Here, we anticipate the long-term relationship between the two time series variables to prevail, regardless of short-term market turbulence.

Therefore, cointegration serves as a valuable tool in statistical analysis, exposing the underlying long-term relationship between two non-stationary and seemingly unrelated time series. This long-term association, difficult to detect when these time series are analyzed independently, can be discovered by combining these individual non-stationary assets in a particular way. This combination is typically done using the Johansen test, yielding a new, combined time series that exhibits stationarity, characterized by a consistent mean and variance over different periods. Alternatively, the Engle-Granger test can be employed to generate a spread series from the residuals of a linear regression model between the two assets.

Figure 8-3 illustrates the process of cointegration and strategy formulation. The purpose of cointegration is to convert individual non-stationary time series data into a combined stationary series, which can be achieved via the Johansen test with a linear

combination, the Engle-Granger test via a linear regression model, or other test procedures. We would then derive another series called the spread to indicate the extent of short-term fluctuation from the long-term equilibrium relationship. The spread is used to generate trading signals in the form of entry and exit points based on the extent of deviation at each time point, with the help of entry and exit thresholds defined in advance.

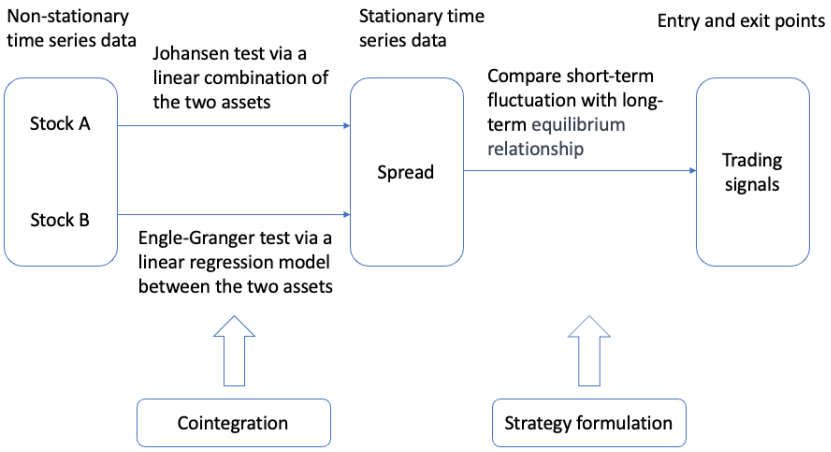

Figure 8-3 Illustrating the process of cointegration using different tests and strategy formulation to generate trading signals.

The next section covers a more in-depth discussion on stationarity.

Stationarity

Stock prices are time series data. A stationary time series is a time series where the statistical properties of the series, including the mean, variance, and covariance at different time points, are constant and do not change over time. A stationary time series is thus characterized by a lack of observable trends or cycles in the data.

Let us take the normal distribution as an example. A normal distribution $y = f(x; \mu, \sigma)$ is a probability density function that maps an input x to a probability output y, assuming a fixed set of parameters: the mean μ as the central tendency and standard deviation σ as the average deviation from the mean. The specific form of the probability distribution is as follows.

$$y = f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

A widely used normal distribution is the standard normal, specifying $\mu = 0$ and $\sigma = 1$. The resulting probability density function is:

$$y = f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

We can generate random samples following this specific using the `random.normal()` function from NumPy. In Listing 8-1, we define a function

`generate_normal_sample()` that generates a normally distributed random sample by passing in the input parameter μ and σ in a list.

Listing 8-1. Generating normal samples

```
# generate random samples from normal distribution
def generate_normal_sample(params):
    """
    input: params, including mean in params[0] and standard
deviation in params[1]
    output: a random sample from the normal distribution
parameterized by the input
    """
    mean = params[0]
    sd = params[1]
    return np.random.normal(mean, sd)
```

Now we generate a sample by specifying a standard normal distribution.

```
# generate sample from standard norml
>>> print(generate_normal_sample([0,1]))
0.09120471661981977
```

To see the impact on the samples generated from a non-stationary distribution, we will specify three different non-stationary distributions. Specifically, we will generate 100 samples that follow a distribution with either an increasing mean or standard deviation. Listing 8-2 performs the random sampling for 100 rounds and compares them with the samples from the standard normal distribution.

Listing 8-2. Generating samples from stationary and non-stationary normal distributions

```
# generate 100 random samples for both stationary and non-stationary
distribution
T = 100
stationary_list, nonstationary_list1, nonstationary_list2 = [], [],
[]

for i in range(T):
    # generate a stationary sample and append to list
    stationary_list.append(generate_normal_sample([0,1]))
    # generate a non-stationary sample with an increasing mean and
append to list
    nonstationary_list1.append(generate_normal_sample([i,1]))
    # # generate a non-stationary sample with an increasing mean and
sd and append to list

nonstationary_list2.append(generate_normal_sample([i,np.sqrt(i)]))
```

```
x = range(T)
# plot the lists as line plots with labels for each line
plt.plot(x, stationary_list, label='Stationary')
plt.plot(x, nonstationary_list1, label='Non-stationary with
increasing mean')
plt.plot(x, nonstationary_list2, label='Non-stationary with
increasing mean and sd')

# set the axis labels
plt.xlabel('Sample index')
plt.ylabel('Sample value')
# add a legend
plt.legend()
# show the plot
plt.show()
```

Running the codes generates Figure 8-4, where the impact of a changing mean and standard deviation becomes more pronounced as we increase the magnitude in later rounds.

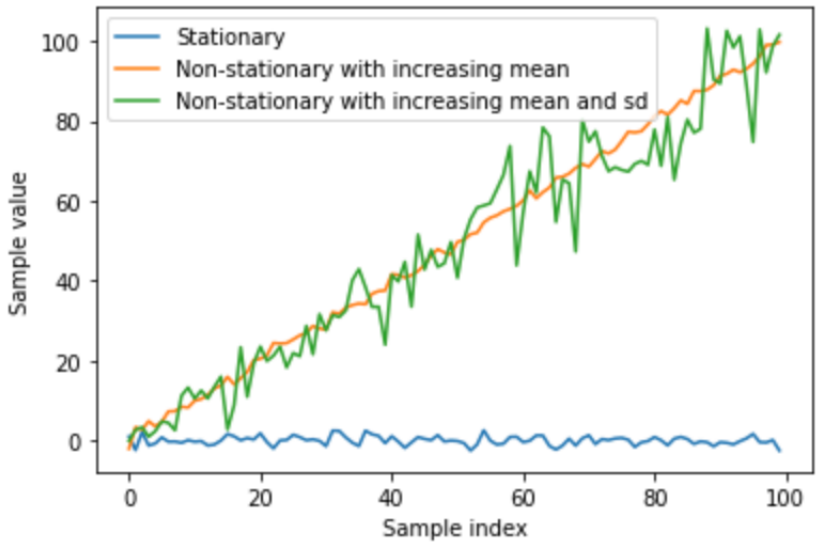

Figure 8-4 Generating normally distributed random samples from non-stationary distributions with different parameter specifications.

Note that we can use the Augmented Dickey-Fuller (ADF) test to check if a series a stationary. The function `stationarity_test()` defined below accepts two inputs: the time series to be tested for stationarity, and the significant level used to compare with the p-value and determine the statistical significance. Note that the p-value is accessed as the second argument from the test result object using the `adfuller()` function. This is shown in Listing 8-3.

Listing 8-3. Testing stationarity of a time series

```
# test for stationarity
def stationarity_test(x, threshold=0.05):
    """
    input:
      x: a list of scalar values
      threshold: significance level
    output: print out message on stationarity
    """
    pvalue = adfuller(x)[1]
    if pvalue < threshold:
        return 'p-value is ' + str(pvalue) + '. The series is likely
stationary.'
    else:
        return 'p-value is ' + str(pvalue) + '. The series is likely
non-stationary.'
```

Let us apply this function to the previous time series data. The result shows that the ADF is able to differentiate if a time series is stationary (with fixed parameters) based on a preset significance level.

```
>>> print(stationarity_test(stationary_list))
>>> print(stationarity_test(nonstationary_list1))
>>> print(stationarity_test(nonstationary_list2))
p-value is 1.2718058919122438e-12. The series is likely stationary.
p-value is 0.9925665941220737. The series is likely non-stationary.
p-value is 0.9120355459829741. The series is likely non-stationary.
```

Let us look at a concrete example of how to test for cointegration between two stocks.

Test for cointegration

This section provides an example of performing the cointegration test using the Engle-Granger two-step method. Here's a general overview of the steps involved:

- Estimate the coefficients of the linear regression model between one stock (as the dependent variable) and the other stock (as the independent variable) using ordinary least squares (OLS).
- Calculate the residuals from the linear regression model.
- Test the residuals for stationarity using a unit root test, such as the augmented Dickey-Fuller (ADF) test.
- If the residuals are stationary, the two stocks are cointegrated. If the residuals are non-stationary, the two stocks are not cointegrated.

Let us illustrate the procedure using two stocks: Google and Microsoft. Listing 8-4 imports necessary packages and download the daily stock prices for the whole year of 2022. We will use the adjusted closing price for the cointegration test.

Listing 8-4. Importing packages and downloading stock data

```
import os
import random
import numpy as np
import yfinance as yf
import pandas as pd
from statsmodels.tsa.stattools import adfuller
from statsmodels.regression.linear_model import OLS
import statsmodels.api as sm
from matplotlib import pyplot as plt
%matplotlib inline

SEED = 8
random.seed(SEED)
np.random.seed(SEED)

# download data from yfinance
start_date  = "2022-01-01"
end_date  = "2022-12-31"
stocks = ['GOOG','MSFT']
df = yf.download(stocks, start=start_date, end=end_date)['Adj
Close']
>>> df.head()
            GOOG       MSFT
Date
2022-01-03 145.074493 330.813873
2022-01-04 144.416504 325.141357
2022-01-05 137.653503 312.659882
2022-01-06 137.550995 310.189301
2022-01-07 137.004501 310.347382
```

Now we dig into the linear regression model between these two stocks. We will treat Google stock as the (only) independent variable, and Microsoft stock as the dependent variable to be predicted. The model assumes the following form:

$$y = \beta_0 + \beta_1 x + \epsilon$$

where β_0 denotes the intercept and β_1 is the slope of the linear line fitted between these two stocks. ϵ represents the random noise that is not modeled by the predictor x. Note that we are assuming a linear relationship between x and y, which is unlikely to be the case in a real-world environment. Another name for ϵ is the residual, which is interpreted as the (vertical) distance between the predicted value $\beta_0 + \beta_1 x$ and the target value y. That is, $\epsilon = y - (\beta_0 + \beta_1 x)$.

Our focus would then shift to these residuals, with the intention of assessing if the residual time series would be stationary. Let us first obtain the residuals from the linear regression model.

In Listing 8-5, we assign the first stock as the target variable `Y` and the second stock as the predictor variable `X`. We then use the `add_constant()` function to add a column of ones to the `X` variable, which can also be considered as the bias trick to incorporate the intercept term β_0. Next, we construct a linear regression model object using the `OLS()` function, perform learning by invoking the `fit()` function, and calculate the residuals as the difference between the target values and the predicted values, obtained via the `predict()` method.

Listing 8-5. Extracting residuals from OLS

```
# build linear regression model
# Extract prices for two stocks of interest
# target var: Y; predictor: X
Y = df[stocks[0]]
X = df[stocks[1]]

# estimate linear regression coefficients of stock1 on stock2
X_with_constant = sm.add_constant(X)
model = OLS(Y, X_with_constant).fit()
residuals = Y - model.predict()
```

The model object is essentially a collection of the model weights (also called parameters) and the architecture that governs how the data flow from the input to the output. Let us access the model weights.

```
# access model weights
>>> print(model.params)
const   -47.680218
MSFT      0.610303
dtype: float64
```

We have two parameters in the model: `const` corresponding to β_0 , and `MSFT` corresponding to β_1.

Besides using the `predict()` method to obtain the predicted values, we can also construct the explicit expression for the predictions and calculate them manually. That is, we can calculate the predicted values $\{\hat{y}\}_{i=1}^{N}$ as follows.

$$\hat{y_i} = \beta_0 + \beta_1 x_i, i \in \{1, \dots, N\}$$

The following code snippet implements this expression and calculates the model predictions manually. We also check if the manually calculated residuals are equal to the previous values using the `equals()` function.

```
# alternative approach
residuals2 = Y - (model.params['const'] + model.params[stocks[1]] *
X)
```

```
# check if both residuals are the same
print(residuals.equals(residuals2))
```

Lastly, we test the stationarity of the residual series, again using the augmented Dickey-Fuller (ADF) test. The test can be performed using the `adfuller()` function from the `statsmodels` package. There are two metrics that are relevant to every statistical test: the test statistic and the p-value. Both metrics convey the same information on the statistical significance of the underlying hypothesis, with the p-value being a standardized and, thus, more interpretable metric. A widely used threshold (also called the significance level) is 5% for the p-value. That is, if the resulting p-value from a statistical test is less than 5%, we can safely (up to a confidence level of 95%) reject the null hypothesis in favor of the alternative hypothesis. If the p-value is greater than 5%, we fail to reject the null hypothesis and conclude that the two stocks are not cointegrated.

The null hypothesis often represents the status quo. In the case of the cointegration testing using the Engle-Granger test, the null hypothesis is that the two stocks are not cointegrated. That is, the historical prices do not exhibit a linear relationship in the long run. The alternative hypothesis is that the two stocks are cointegrated, as exhibited by a linear relationship between the two and a stationary residual series.

Now let us carry out the ADF test and use the result to determine if these two stocks are cointegrated using a significance level of 5%. In Listing 8-6, we apply the `adfuller()` function to the prediction residuals and print out the test statistic and p-value. This is followed by an if-else statement to determine if we have enough confidence to reject the null hypothesis and claim that the two stocks are cointegrated.

Listing 8-6. Testing stationarity of the residuals

```
# test stationarity of the residuals
adf_test = adfuller(residuals)
print(f"ADF test statistic: {adf_test[0]}")
print(f"p-value: {adf_test[1]}")

if adf_test[1] < 0.05:
    print("The two stocks are cointegrated.")
else:
    print("The two stocks are not cointegrated.")
ADF test statistic: -3.179800920038961
p-value: 0.021184058997635733
The two stocks are cointegrated.
```

The result suggests that Google and Microsoft stocks are cointegrated due to a small p-value of 2%. Indeed, based on our previous analysis of calculating the max drawdown, Google and Microsoft stock prices generally tend to move together. However, with the introduction of ChatGPT in Bing search, the overall picture may start to change. Such cointegration (co-movement) may gradually weaken as the tool gives everything for

Microsoft to win (due to a small revenue from web search) and for Google to lose (majority revenue comes from web search).

Next, we touch upon another closely related but different statistical concept: correlation.

Correlation and cointegration

Both correlation and cointegration are important statistical measures used to analyze the relationship between two time series datasets. Correlation quantifies the degree of linear association between two time series. In essence, it reveals whether the two variables increase or decrease in tandem, and the strength of this relationship. The correlation coefficient can vary between -1 and 1. A coefficient of 1 denotes a perfect positive linear relationship, -1 signifies a perfect negative linear relationship, while 0 suggests the absence of any linear relationship.

In contrast, cointegration is concerned with the long-term equilibrium relationship between two potentially non-stationary time series. If two time series are cointegrated, it signifies that they share a common long-term trend, regardless of their short-term variations. Consequently, while the two time series may not exhibit short-term linear correlation, they can display a long-term stationary pattern when suitably combined. This enables analysts to uncover persistent relationships masked by transitory market volatility.

The code snippet below provides an example of two correlated time series that are not cointegrated. We first sample two series of 100 random values following normal distributions with a different mean and the same variance. This is followed up by a cumulative summation operation stored as a Pandas Series object. Finally, we plot both series as lines after combining them horizontally in a DataFrame and calling the plot() function.

```
np.random.seed(123)
X = np.random.normal(1, 1, 100)
Y = np.random.normal(2, 1, 100)

X = pd.Series(np.cumsum(X), name='X')
Y = pd.Series(np.cumsum(Y), name='Y')

pd.concat([X, Y], axis=1).plot()s
```

Running the codes generates Figure 8-5. Series Y has a higher drift than series X as designed, and also exhibits a high degree of correlation (or co-movement) across the whole history of 100 points.

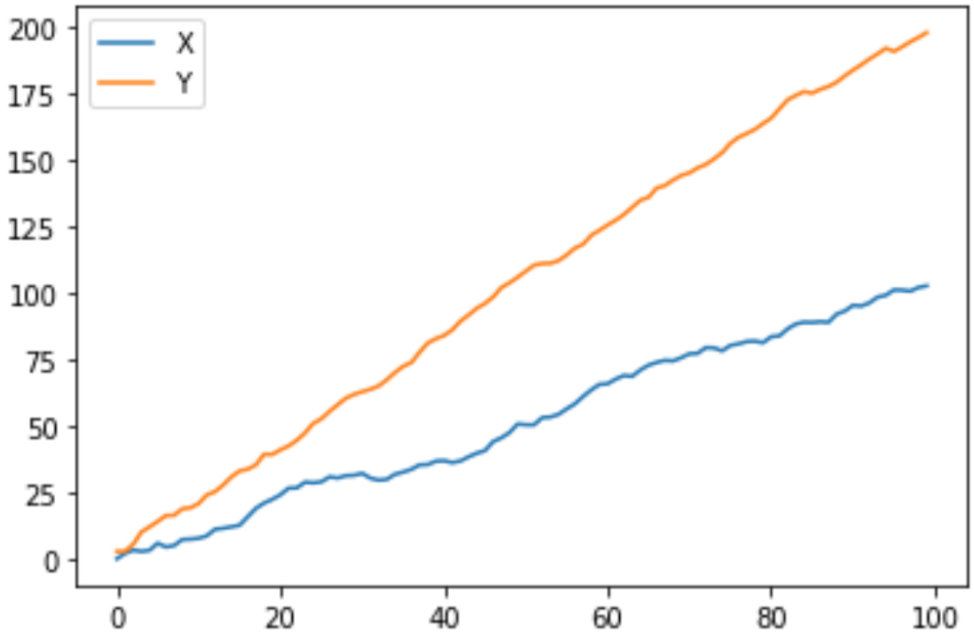

Figure 8-5 Illustrating the evolution of two series that are highly correlated but not cointegrated.

Let us calculate the exact correlation coefficient and cointegration p-value. In the following code snippet, we call the `corr()` method to obtain the correlation of X with Y, and use the `coint()` function from the `statsmodels` package to perform the cointegration test and retrieve the resulting p-value. The `coint()` function performs the augmented Engle-Granger two-step cointegration test, similar to how to manually carry out the two-step process earlier. The result shows that these two series are highly correlated but not cointegrated.

```
from statsmodels.tsa.stattools import coint
# calculate the correlation coefficeint
>>> print('Correlation: ' + str(X.corr(Y)))
# perform in cointegration test
score, pvalue, _ = coint(X,Y)
>>> print('Cointegration test p-value: ' + str(pvalue))
Correlation: 0.994833254077976
Cointegration test p-value: 0.17830098966789126
```

In the next section, we dive deep into the implementation of the pairs trading strategy.

Implementing the pairs trading strategy

As a market-neutral trading strategy, pairs trading identifies two cointegrated stocks based on a specific statistical test procedure using historical data. It takes a long and a short position in these two stocks simultaneously. Therefore, no matter whether the market moves up or down for these two stocks, there is no impact on the pairs trading strategy, so as long their relative

spread remains the same. Instead, the strategy monitors the spread between the two stocks, which should remain relatively constant over time, and makes a move in case of short-term price movements based on preset thresholds.

Let us first download the stock price data. We will focus on a few stock symbols of major tech giants: Google, Microsoft, Apple, Tesla, Meta, and Netflix. The following code snippet downloads the historical stock prices for the full year of 2022 and extracts the adjusted closing prices to the `df` variable.

```
# download data from yfinance
stocks = ['GOOG','MSFT','AAPL','TSLA','META','NFLX']
df = yf.download(stocks, start=start_date, end=end_date)['Adj
Close']
```

Next, we analyze each unique pair of stocks and perform the cointegration test to look for those with a long-term equilibrium relationship.

Identifying cointegrated pairs of stocks

There is a total of six stocks in our search space, leading to a total of $C_6^2 = 15$. Generating the list of unique pairs of stocks can be performed via the `combinations()` function from `itertools` package, as shown in Listing 8-7.

Listing 8-7. Generating all unique pairs of stocks

```
from itertools import combinations

# get all pairs of stocks
stock_pairs = list(combinations(df.columns, 2))
>>> stock_pairs
[('AAPL', 'GOOG'),
 ('AAPL', 'META'),
 ('AAPL', 'MSFT'),
 ('AAPL', 'NFLX'),
 ('AAPL', 'TSLA'),
 ('GOOG', 'META'),
 ('GOOG', 'MSFT'),
 ('GOOG', 'NFLX'),
 ('GOOG', 'TSLA'),
 ('META', 'MSFT'),
 ('META', 'NFLX'),
 ('META', 'TSLA'),
 ('MSFT', 'NFLX'),
 ('MSFT', 'TSLA'),
 ('NFLX', 'TSLA')]
```

These 15 unique pairs of stocks are stored as tuples in a list. Each tuple will go through the cointegration test in the following section.

Testing pairwise cointegration

In Listing 8-8, we loop through each pair of stocks and perform the Engle-Granger test using the `coint()` function. For each unique pair of stocks, we first extract the corresponding DataFrame via subsetting by column names, and then perform the cointegration test using the two series to obtain the test score and p-value. We will then compare the p-value with a preset threshold and print out the result to assess if the test result is statistically significant.

Listing 8-8. Performing cointegration test for each unique pair of stocks

```
threshold = 0.1
# run Engle-Granger test for cointegration on each pair of stocks
for pair in stock_pairs:
    # subset df based on current pair of stocks
    df2 = df[list(pair)]
    # perform test for the current pair of stocks
    score, pvalue, _ = coint(df2.values[:,0], df2.values[:,1])
    # check if the current pair of stocks is cointegrated
    if pvalue < threshold:
        print(pair, 'are cointegrated')
    else:
        print(pair, 'are not cointegrated')
```

Note that the threshold is set as 10% instead of 5% as before, since the test would show no cointegrated pair of stocks when setting the threshold as the latter. As it turns out, the `coint()` function is slightly different from our manual implementation of the test procedure earlier. For example, the order of the time series assumed by the `coint()` function may not be the same.

Running the code generates the following result.

```
('AAPL', 'GOOG') are not cointegrated
('AAPL', 'META') are not cointegrated
('AAPL', 'MSFT') are not cointegrated
('AAPL', 'NFLX') are not cointegrated
('AAPL', 'TSLA') are not cointegrated
('GOOG', 'META') are not cointegrated
('GOOG', 'MSFT') are cointegrated
('GOOG', 'NFLX') are not cointegrated
('GOOG', 'TSLA') are not cointegrated
('META', 'MSFT') are not cointegrated
('META', 'NFLX') are not cointegrated
('META', 'TSLA') are not cointegrated
('MSFT', 'NFLX') are not cointegrated
```

```
('MSFT', 'TSLA') are not cointegrated
('NFLX', 'TSLA') are not cointegrated
```

It turns out that only Google and Microsoft stock prices are cointegrated using the 10% threshold on the significance level. These two stocks will be the focus of our pairs trading strategy in the following strategy, starting by identifying the stationary spread between the two stocks.

Obtaining the spread

As introduced earlier, the spread is a time series derived from the historical data of the two stocks in the pairs trading strategy. There are many ways to calculate the spread, and we will go with the one employed in the cointegration test procedure. Specifically, we define the spread as the residuals from the linear regression model between the two stocks. If they pass the cointegration test, we have confidence (up to 90% confidence level) that these two stocks, when linearly combined, generate a stationary time series in the spread.

Listing 8-9 generates the spread time series and visualizes it in a line plot. As before, we first extract the predictor `X` and target `Y`, apply the bias trick by adding a column of constant ones to `X`, run the linear regression model, and finally obtain the spread as the residual between the target and the prediction.

Listing 8-9. Calculating the spread

```
# calculate the spread for GOOG and MSFT
Y = df["GOOG"]
X = df["MSFT"]
# estimate linear regression coefficients
X_with_constant = sm.add_constant(X)
model = OLS(Y, X_with_constant).fit()
# obtain the spread as the residuals
spread = Y - model.predict()
spread.plot(figsize=(12,6))
```

Running the code generates Figure 8-6. The spread now appears as white noise, that is, following a normally distributed Gaussian distribution. Since different stocks have different scales of spread, it would be recommended to standardize them into the same scalar for ease of comparison and strategy formulation. The next section covers the conversion process that turns the spread into z-scores.

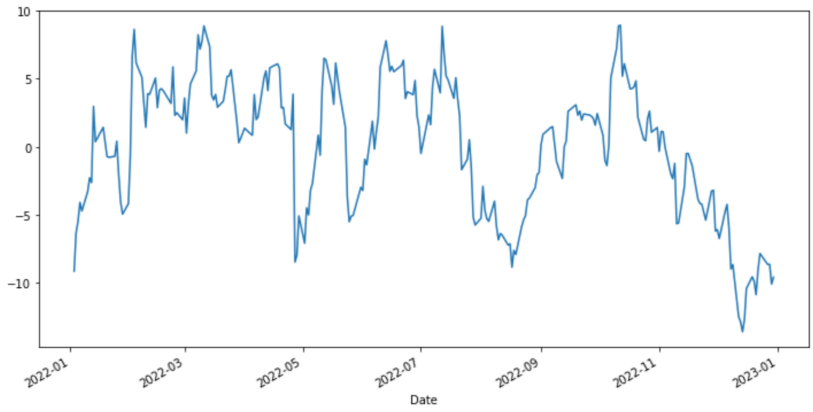

Figure 8-6 Visualizing the spread as the residuals of the linear regression model.

Converting to z-scores

A z-score is a measure of how many standard deviations the daily spread is from its mean. It is a standardized score that we can use to compare across different distributions. Denote x as the original observation. The z-score is calculated as follows:

$$z = \frac{x - \mu}{\sigma}$$

where μ and σ denote the mean and standard deviation of the time series, respectively.

Therefore, the magnitude of the z-score indicates how far away the current observation deviates from the mean in terms of the unit of standard deviations, and the sign of the z-score suggests whether the deviation is above (a positive z-score) or below (a negative z-score) the mean.

For example, assume a distribution with a mean of 10 and a standard deviation of 2. If an observation is valued at 8, the z-score for this observation would be $\frac{10-8}{2} = 1$. In other words, this observation is one standard deviation away from the mean of the distribution.

The z-score is often used to assess the statistical significance of observation in hypothesis testing. A z-score of greater than or equal to 1.96 (or, smaller than or equal to -1.96) corresponds to a p-value of 0.05 or less, which is a common threshold for assessing the statistical significance.

In Listing 8-10, we visualize the probability density function (PDF) of a standard normal distribution with a mean of 0 and a standard deviation of 1. We first generate a list of equally-spaced input values as the z-scores using the `np.linspace()` function and obtain the corresponding probabilities in the PDF of standard normal distribution using the `norm.pdf()` function with a location parameter of 0 (corresponding to the mean) and scale of 1 (corresponding to the standard deviation). We also shade the areas before -1.96 and after 1.96, where a z-score of 1.96 corresponds to a 95% significance level in a statistical test. In other words, z-scores greater than or equal to 1.96 accounts for 5% of the total probability, and z-scores lower than or equal to -1.96 account for 5% as well.

Listing 8-10. Calculating the z-score

```
# illustrate z score by generating a standard normal distribution
with mu 0 and sd 1
from scipy.stats import norm
# input: unbounded scalar, assumed to be in the range of [-5,-5] in
this case
x = np.linspace(-5, 5, 100)
# output: probability between 0 and 1
y = norm.pdf(x, loc=0, scale=1)
# set up the plot
fig, ax = plt.subplots()
# plot the pdf of normal distribution
ax.plot(x, y)
# shade the area corresponding to a z-score of >=1.96 and <=-1.96
z_critical = 1.96
x_shade = np.linspace(z_critical, 5, 100)
y_shade = norm.pdf(x_shade, loc=0, scale=1)
ax.fill_between(x_shade, y_shade, color='red', alpha=0.3)
z_critical2 = -1.96
x_shade2 = np.linspace(-5, z_critical2, 100)
y_shade2 = norm.pdf(x_shade2, loc=0, scale=1)
ax.fill_between(x_shade2, y_shade2, color='red', alpha=0.3)
# add labels and a title
ax.set_xlabel('Z-score')
ax.set_ylabel('Probability density')
# add a vertical line to indicate the z-score of 1.96 and -1.96
ax.axvline(x=z_critical, linestyle='--', color='red')
ax.axvline(x=z_critical2, linestyle='--', color='red')
# display the plot
plt.show()
```

Running the codes generates Figure 8-7.

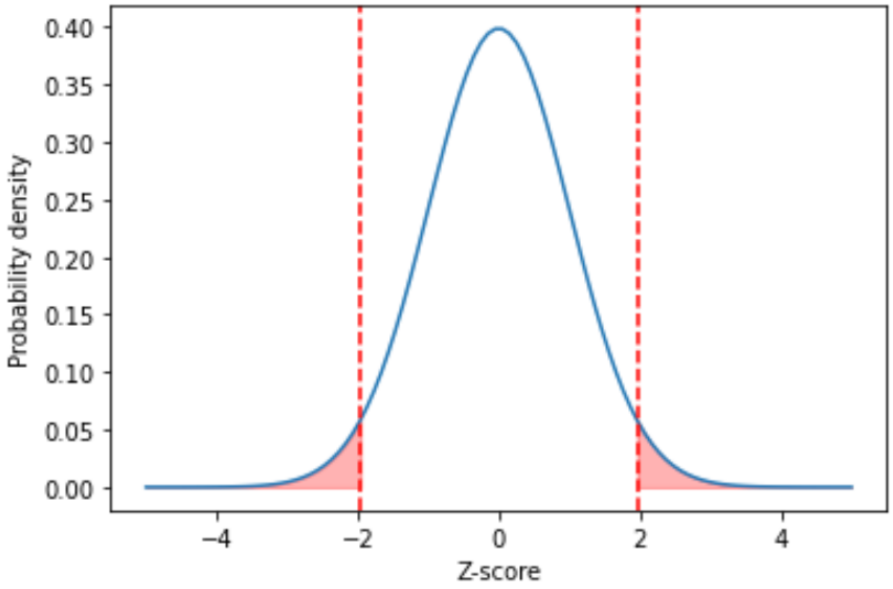

Figure 8-7 Visualizing the probability density function of a standard normal distribution, with the 5% significance level shaded at both the left and right sides.

In the context of hypothesis testing, the shaded area represents the probability of observing a z-score greater than 1.96 under the null hypothesis. Performing the statistical test would give us a z-score. If the z-score is above 1.96 or below -1.96 in a one-sided test, we would reject the null hypothesis in favor of the alternative hypothesis at the 0.05 significance level, since the probability of observing the phenomenon under the null hypothesis would simply be too small.

In summary, we use the z-score as a standardized score to measure how many standard deviations an observation is from the mean of a distribution. It is used in hypothesis testing to determine the statistical significance of an observation, that is, the probability of an event happening under the null hypothesis. The significance level is often set at 0.05. We can use the z-score to calculate the probability of observing a value as extreme as the observation under the null hypothesis. Finally, we make a decision on whether to reject or fail to reject the null hypothesis.

Coming back to our running example. Since stock prices are often volatile, we switch to the moving average approach to derive the running mean and standard deviation. That is, each daily spread would have a corresponding running mean and standard deviation based on the collection of spreads in the rolling window. In Listing 9-10, we derive the running mean and standard deviation using a window size of 10, and apply the transformation to derive the resulting z-scores as the standardized spread.

Code listing 9-10. Converting to z-score based on moving averages

```
# convert to z score
# z-score is a measure of how many standard deviations the spread is
from its mean
# derive mean and sd using a moving window
```

```
window_size = 10
spread_mean = spread.rolling(window=window_size).mean()
spread_std = spread.rolling(window=window_size).std()
zscore = (spread - spread_mean) / spread_std
zscore.plot(figsize=(12,6))
```

Running the codes generates Figure 8-8, where the standardized spreads now look more normally distributed as white noise.

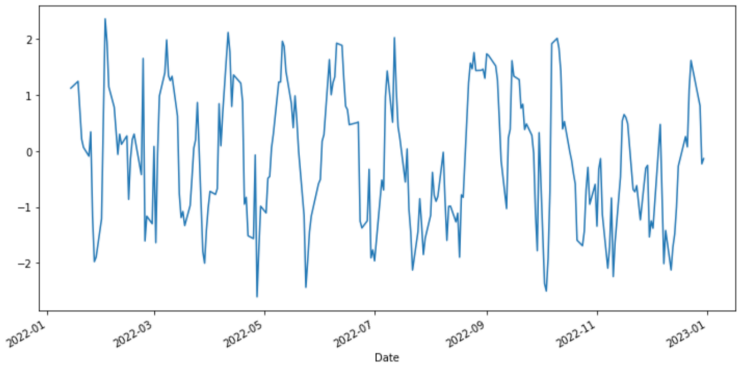

Figure 8-8 Visualizing the z-scores after standardizing the spreads using the running mean and standard deviation.

Since we used a window size of 10, the first 9 observations will appear as NA in the moving average series. Let us get rid of the initial NA values by first identifying the first valid index using the `first_valid_index()` function and then subsetting the z-score series, as shown in the following code.

```
# remove initial days with NA
first_valid_idx = zscore.first_valid_index()
zscore = zscore[first_valid_idx:]
>>> zscore
Date
2022-01-14    1.123748
2022-01-18    1.245480
2022-01-19    0.742031
2022-01-20    0.211878
2022-01-21    0.064889
                 ...
2022-12-23    1.618937
2022-12-27    0.977235
2022-12-28    0.807607
2022-12-29   -0.230086
2022-12-30   -0.137035
Name: GOOG, Length: 242, dtype: float64
```

The next section formulates the trading strategy using the z-scores.

Formulating the trading strategy

As introduced earlier, the pairs trading strategy utilizes the z-scores to generate trading signals in the face of short-term fluctuations in the spread, taking long and short positions in two cointegrated assets and profiting from the long-term mean reversion of the spread.

The trading signals are generated when the z-score obtained from the previous section crosses over a specific threshold. For example, we can long the first stock and short the second stock when the z-score is below -2, meaning that the spread is more negative than usual, and there is a good chance that the spread will revert back to its mean in the long run. Similarly, we can short the first stock and long the second stock when the z-score is above 2, suggesting that the spread is more positive than usual and there is a good chance that the spread will go back to its mean. These constitute our entry signals.

On the other hand, when we are in an open position, the stock may move in an adverse direction in a very small amount of time. To protect our profit and stop the loss, we can place an exit signal that serves as a stop-loss order. For example, assume we entered a long position when the z-score was below -2 in the previous step. We can set up another threshold to exit the position when the z-score returns to a small value, say -1. Crossing this threshold indicates that the spread has reverted back to its mean.

The following list summarizes the formulation of trading signals for entering and exiting the long and short positions.

- Long entry: Enter a long position in the first stock when the z-score is below a preset negative threshold value (say -2).
- Long exit: Exit the long position in the first stock when the z-score crosses above another preset negative threshold value (say -1).
- Short entry: Enter a short position in the second stock when the z-score is above a preset positive threshold value (say 2).
- Short exit: Exit the short position in the second stock when the z-score crosses below another preset negative threshold value (say 1).

To manage these four types of signals in implementation, we could maintain a Pandas Series object for each stock, where each value is either 1 (for long), -1 (for short), or 0 (for exit position). To simplify the process, we also assume that the long and short positions for each stock are also entered and exited together. In other words, upon entering a long position for one stock, we would enter a short position in the other stock at the same time.

Figure 8-9 overlays these four trading signals in the previous z-score time series. The outer thresholds 2 and -2 represent entry signals for long and short positions and the inner thresholds 1 and -1 represent the exit signals for existing positions. In between these two thresholds, we simply maintain the current position.

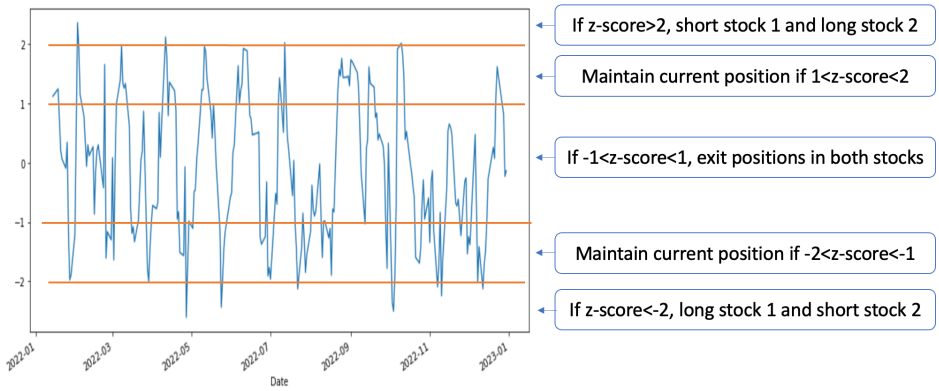

Figure 8-9 Illustrating the process of formulating trading signals based on preset entry and exit thresholds for the z-scores.

In Listing 8-11, we first initialize the entry and exit thresholds, respectively. We create two Pandas Series objects (`stock1_position` and `stock2_position`) to store the daily positions for each stock. Based on the current z-score and present thresholds for entering and exiting long or short positions, we check the daily z-score in a loop and match it to one of the four cases for signal generation based on the following rule:

- Long stock 1 and short stock 2 if the z-score is below -2 and stock 1 has no prior position.
- Short stock 1 and long stock 2 if the z-score is above 2 and stock 2 has no prior position.
- Exit the position in both stock 1 and stock 2 if the z-score is between -1 and 1.
- Maintain the position in both stock 1 and stock 2 for the rest of the cases, that is, the z-score is between -2 and -1, or between 1 and 2.

Listing 8-11. Implementing pairs trading

```
# set the threshold values for entry and exit signals
entry_threshold = 2.0
exit_threshold = 1.0
# initialize the daily positions to be zeros
stock1_position = pd.Series(data=0, index=zscore.index)
stock2_position = pd.Series(data=0, index=zscore.index)
# generate daily entry and exit signals for each stock
for i in range(1, len(zscore)):
    # zscore<-2 and no existing long position for stock 1
    if zscore[i] < -entry_threshold and stock1_position[i-1] == 0:
        stock1_position[i] = 1 # long stock 1
        stock2_position[i] = -1 # short stock 2
    # zscore>2 and no existing short position for stock 2
```

```
    elif zscore[i] > entry_threshold and stock2_position[i-1] == 0:
        stock1_position[i] = -1 # short stock 1
        stock2_position[i] = 1 # long stock 2
    # -1<zscore<1
    elif abs(zscore[i]) < exit_threshold:
        stock1_position[i] = 0 # exit existing position
        stock2_position[i] = 0
    # -2<zscore<-1 or 1<zscore<2
    else:
        stock1_position[i] = stock1_position[i-1] # maintain
existing position
        stock2_position[i] = stock2_position[i-1]
```

We can now calculate the overall profit of the pairs trading strategy. In Listing 8-12, we first obtain the daily percentage changes using the `pct_change()` function for each stock, starting from the index with a valid value. These daily returns will be adjusted according to the position we held from the previous trading day. In other words, multiplying the shifted positions with the daily returns gives the strategy's daily returns for each stock, filling possible NA values with zero. Finally, we add up the daily returns from the two stocks, convert them to 1+R returns, and perform the sequential compounding procedure using the `cumprod()` function to obtain the wealth index.

Listing 8-12. Calculating the cumulative return

```
# Calculate the returns of each stock
stock1_returns = (df["GOOG"][first_valid_idx:].pct_change() *
stock1_position.shift(1)).fillna(0)
stock2_returns = (df["MSFT"][first_valid_idx:].pct_change() *
stock2_position.shift(1)).fillna(0)
# calculate the total returns of the strategy
total_returns = stock1_returns + stock2_returns
cumulative_returns = (1 + total_returns).cumprod()
# plot the cumulative returns
>>> cumulative_returns.plot()
```

Running the codes generates Figure 8-10.

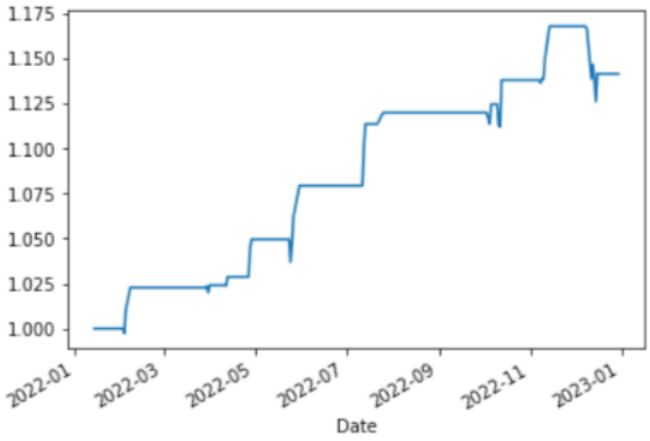

Figure 8-10 Cumulative returns of the pairs trading strategy.

The terminal return, extracted via the following codes, shows that the pairs trading strategy delivers a total of 14.1% profit at the end of the trading year.

Again, this result is subject to more rigorous backtesting in terms of the selection of investment assets, trading periods, and evaluation metrics.

Summary

In this chapter, we covered the concept of statistical arbitrage and hypothesis testing, as well as the implementation details based on the pairs trading strategy. We first walked through the overall process of developing a pairs trading strategy, and introduced new concepts such as cointegration and stationarity. Next, we compared cointegration and correlation, both closely related but drastically different. Last, we introduced a case study on calculating the cumulative return using the pairs trading strategy.

In the next chapter, we will introduce Bayesian optimization, a principled way to search for optimal parameters of a trading strategy.

Exercises

- Evaluate the cointegration of selected stock pairs during bull and bear market periods separately. Do the results vary significantly? If so, discuss possible reasons.
- Implement rolling cointegration tests on a pair of time series data and observe how cointegration status (cointegrated or not) evolves over time.
- For a given pair of stocks, test the stationarity of the spread between them using the ADF test. If the spread is stationary, what does it imply for pairs trading strategy?

- Given the time series data of spreads for a pair of stocks, perform a hypothesis test to check whether the mean of spreads is equal to zero.
- Calculate the z-scores of the spread for different lookback periods (e.g., 30, 60, and 90 days). How does changing the lookback period affect the distribution of z-scores and the performance of your pairs trading strategy?