Chapter 5 Trend Following Strategy

Trend following is a popular investment strategy used in all types of markets, including stocks, bonds, commodities, currencies, and even cryptocurrencies. As its name suggests, this strategy is based on the assumption that prices tend to move in a particular direction (or "trend") over time, thus offering opportunities to capitalize on these movements. At its core, trend following involves analyzing historical price data to identify potential trends. The strategy then recommends taking positions that align with these trends with the expectation that they will continue. For example, if the price of an asset has been steadily rising, a trend follower would typically take a long position, expecting the upward trend to continue. Conversely, if the price has been consistently falling, the trend follower might take a short position, betting that the price will continue to drop.

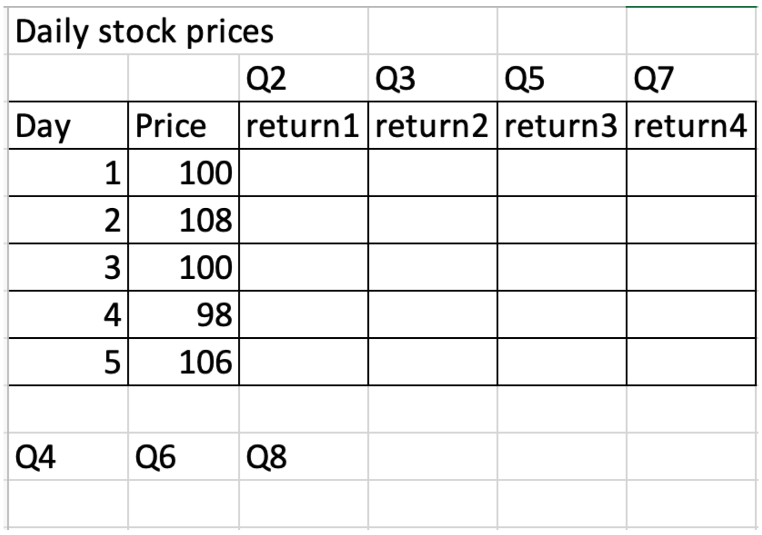
However, like any trading strategy, trend following is not foolproof. Trends can reverse suddenly due to unexpected market events or changes in market sentiment, leading to potential losses. Therefore, trend-following strategies typically include overlaying risk management techniques, such as setting up stop-loss orders, to limit potential losses when the trend reverses.

Trend-following strategies use a variety of technical indicators to identify and confirm trends, such as moving averages, trend lines, and momentum indicators. This chapter introduces the working mechanism of the trend-following strategies using moving averages and then shows its implementation in Python.

Since we will be working with log returns mostly, let us start by going through an example of its calculation process.

Working with log returns

Let us build a further understanding of the logarithmic return (or log return) as we will use it to calculate the stock returns when assessing the trend-following strategy. We start with the following Excel table in Figure 5-1, where we are given a set of dummy stock prices and are asked to answer questions from Q1 to Q9. We detail the questions and answers in the following.



Daily stock prices					
		Q2	Q3	Q5	Q7
Day	Price	return1	return2	return3	return4
1	100				
2	108				
3	100				
4	98				
5	106				
Q4	Q6	Q8			

Figure 5-1 Daily dummy stock prices.

Let us go through each of these nine questions.

Q1: why do we use percentage return?

Answer: Percentage return provides the same scale of comparison. For example, when we have the price data of another stock (stock B) in the range of 1-10, comparing it with the stock price data (stock A) given by the Excel table is difficult when using absolute terms. A $5 increase means more for stock A than stock B. By converting them to the relative percentage terms, we can put both stocks on the scale ruler and measure their performance. Thus using percentage returns, we can accurately compare the performance of these two stocks despite their difference in price levels.

Percentage returns are also useful for comparing the performance of an investment to a benchmark or standard, such as a market index (like the S&P 500 or the Dow Jones Industrial Average). This helps investors to assess how well an investment or a portfolio is performing relative to the broader market or a sector of the market.

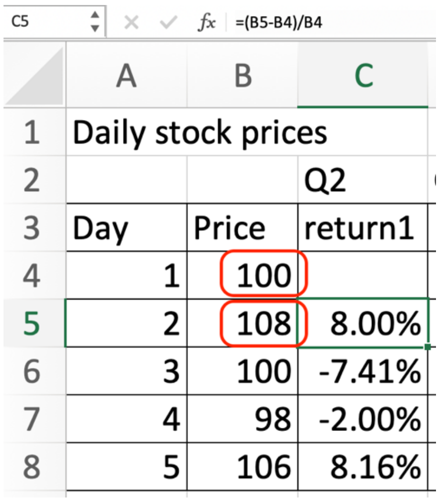
Q2: Calculate single-period percentage return the original way (based on the definition of return).

Answer: The single-period percentage return, also known as the simple return or the holding period return, reflects the percentage change in the value of an investment from one period to the next. It is calculated as:

$$R_{t,t+1} = \frac{S_{t+1} - S_t}{S_t}$$

where $R_{t,t+1}$ is the single-period percentage return from time period t to $t+1$, S_t and S_{t+1} are asset price at the end of period t and $t+1$, respectively. The numerator of the formula, $S_{t+1} - S_t$, calculates the change in the price of the asset from time t to $t+1$. The denominator, S_t, is the price at the beginning of the period, which serves as the baseline for measuring the relative change. Dividing the price change by the starting price gives the relative change in price, expressed as a percentage, which is the simple return.

Applying the same formula to all cells in column return1 except for day 1 generates the result in Figure 5-2.



C5 fx =(B5-B4)/B4

	A	B	C
1	Daily stock prices		
2			Q2
3	Day	Price	return1
4	1	100	
5	2	108	8.00%
6	3	100	-7.41%
7	4	98	-2.00%
8	5	106	8.16%

Figure 5-2 Calculating the simple returns based on the definition of percentage return.

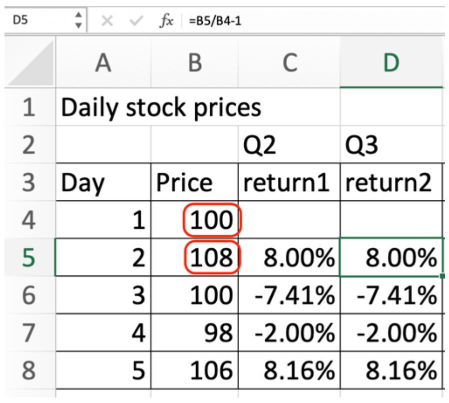
Q3: Calculate the same returns using the 1+R way.

Answer: The 1+R approach to calculating returns is slightly different from the original method but essentially delivers the same result. This approach emphasizes the growth factor of the asset's price from one period to the next, making it easier to understand and interpret. The 1+R approach says that we rewrite the return as:

$$R_{t,t+1} = \frac{S_{t+1}}{S_t} - 1$$

This requires two steps: first, calculate the ratio $\frac{S_{t+1}}{S_t}$ to obtain the so-called 1+R return. This ratio reflects the growth factor of the asset's price from the beginning of the period to the end. If this ratio is greater than 1, it indicates that the asset's price has increased over the period. If it's less than 1, it indicates a decrease in the asset's price. If the ratio equals 1, it means the asset's price hasn't changed.

Next, we would subtract 1 from the 1+R return to convert it to the simple return. This step transforms the growth factor $\frac{S_{t+1}}{S_t}$ into the actual percentage return. Subtracting 1 essentially removes the initial investment from the calculation, leaving only the gained or lost amount relative to the initial investment, which is the return. See Figure 5-3 for an illustration, where the daily returns are the same the in the previous approach.


D5 =B5/B4-1

	A	B	C	D
1	Daily stock prices			
2			Q2	Q3
3	Day	Price	return1	return2
4	1	100		
5	2	108	8.00%	8.00%
6	3	100	-7.41%	-7.41%
7	4	98	-2.00%	-2.00%
8	5	106	8.16%	8.16%

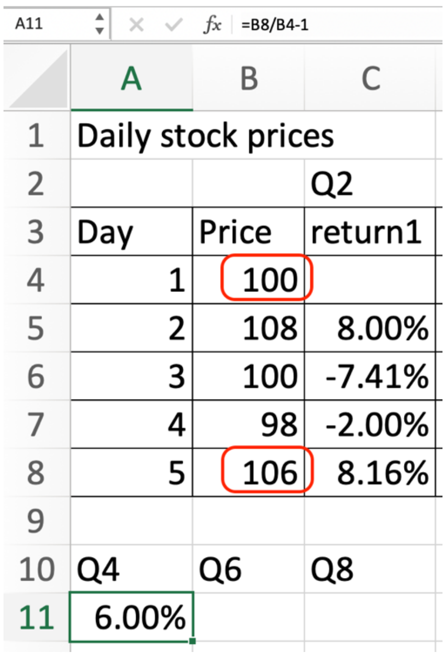
Figure 5-3 Calculating the simple returns based on the 1+R approach.

This 1+R method is often used because it is more intuitive. The growth factor $\frac{S_{t+1}}{S_t}$ easily shows how much the initial investment has grown (or shrunk), and subtracting 1 gives the net growth in percentage terms, which is the simple return. This method is especially useful when dealing with multiple time periods, as growth factors can simply be multiplied together to calculate the cumulative growth factor over several periods.

Q4: what is the terminal return from day 1 to day 5 without compounding?

Answer: The terminal return is the total return on an investment over a given period of time. It's a measure of the total gain or loss experienced by an investment from the start of the investment period to the end, without considering any compounding effect over the period.

To calculate the terminal return without involving the compounding process, we would resort to $R_{1,5} = \frac{S_5 - S_1}{S_1} = \frac{S_5}{S_1} - 1$, where the second formula first calculates the ratio of the asset's price on day 5 to its price on day 1 (which reflects the overall growth factor) and then subtracts 1 to convert the growth factor into a terminal return. See Figure 5-4 for an illustration.


A11 | =B8/B4-1

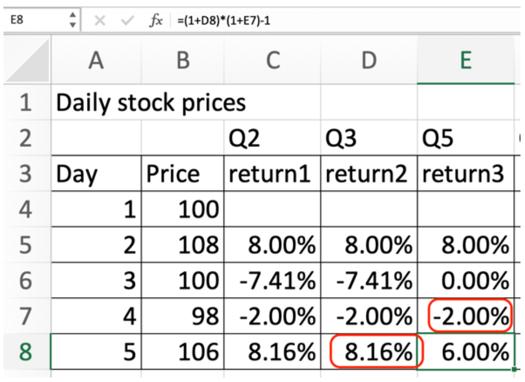
	A	B	C
1	Daily stock prices		
2			Q2
3	Day	Price	return1
4	1	100	
5	2	108	8.00%
6	3	100	-7.41%
7	4	98	-2.00%
8	5	106	8.16%
9			
10	Q4	Q6	Q8
11	6.00%		

Figure 5-4 Calculating the terminal return without compounding.

Q5: what is the terminal return from day 1 to day 5 with compounding? Is it equal to the result in Q4?

Answer: Compounding returns is an important concept in finance. It reflects the fact that not only your initial investment earns a return, but also the returns from previous periods. This leads to exponential growth over time, given a positive return rate.

We will fill in the “return3” column, where each cell is a product between the 1+R return of the current period and the cumulative 1+R return of the previous period, offset by one. For the first period (from day 1 to day 2), the "return3" value would be just the "1 + R" return for this period. See Figure 5-5 for an illustration.


E8 | =(1+D8)*(1+E7)-1

	A	B	C	D	E
1	Daily stock prices				
2			Q2	Q3	Q5
3	Day	Price	return1	return2	return3
4	1	100			
5	2	108	8.00%	8.00%	8.00%
6	3	100	-7.41%	-7.41%	0.00%
7	4	98	-2.00%	-2.00%	-2.00%
8	5	106	8.16%	8.16%	6.00%

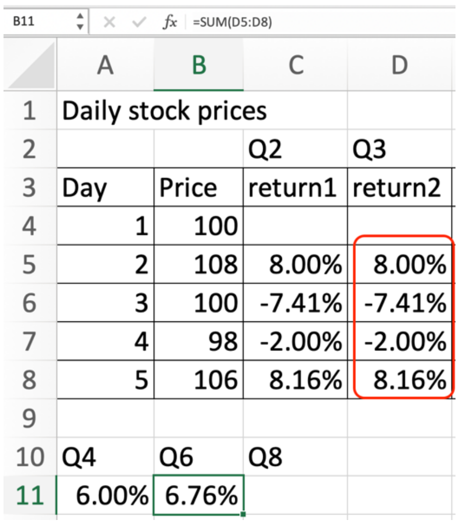
Figure 5-5 Calculating the terminal return using compounding.

As it turns out, the terminal return is 6%, which is the same as previously calculated.

Q6: Sum up the single-period returns in Q3. Is it equal to the result in Q4?

Answer: The result shows that it is different from 6%. In general, adding up single-period returns can lead to incorrect conclusions about the overall return on investment. The sum of the single-period returns is not equal to the terminal return (from Q4) because this approach overlooks the effect of compounding. In other words, by simply summing up single-period returns, we are effectively treating each period's return as if it was independent and earned on the initial investment amount, disregarding the fact that the investment grows with each period due to the returns earned in the prior periods. This is why we see a difference between the summed single-period returns and the terminal return calculated through the correct method that takes into account the compounding effect.

The principle of compounding acknowledges that returns accumulate over time, meaning the returns earned in one period are reinvested and can generate further returns in subsequent periods. So, while the sum of single-period returns might provide a rough estimate of the total return, it is not a correct measure, especially when the time span is long, or the return rate is high. Instead, the appropriate way to calculate the total return over multiple periods is to use the concept of compound returns, which considers both the initial investment and the reinvestment of returns. It is thus important to follow the sequential compounding process when calculating the terminal return. See Figure 5-6 for an illustration.


B11 =SUM(D5:D8)

	A	B	C	D
1	Daily stock prices			
2			Q2	Q3
3	Day	Price	return1	return2
4	1	100		
5	2	108	8.00%	8.00%
6	3	100	-7.41%	-7.41%
7	4	98	-2.00%	-2.00%
8	5	106	8.16%	8.16%
9				
10	Q4	Q6	Q8	
11	6.00%	6.76%		

Figure 5-6 Summing up all single-period returns.

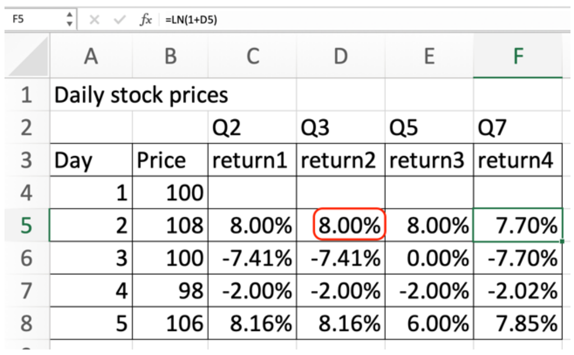
Q7: Calculate the log return for each period.

Answer: The logarithmic return, or continuously compounded return, is another method of calculating returns that can simplify various calculations in finance. This method uses the natural logarithm (log) to express the rate of return, which is derived from the relative changes in price.

To calculate the log return for each period, we can use the formula:

$$\text{log\_return} = \ln \frac{S_{t+1}}{S_t}$$

Here, S_{t+1} and S_t represent the asset price at the future time $t+1$ and the current time t, respectively, and $\ln$ denotes the natural logarithm. See Figure 5-7 for an illustration.



F5 =LN(1+D5)

	A	B	C	D	E	F
1	Daily stock prices					
2			Q2	Q3	Q5	Q7
3	Day	Price	return1	return2	return3	return4
4	1	100				
5	2	108	8.00%	8.00%	8.00%	7.70%
6	3	100	-7.41%	-7.41%	0.00%	-7.70%
7	4	98	-2.00%	-2.00%	-2.00%	-2.02%
8	5	106	8.16%	8.16%	6.00%	7.85%

Figure 5-7 Calculating the log returns of each period.

For instance, if we have the price data in a sequence, we can compute the log return for each period using this formula. Note that the log return is a good approximation for small returns, and it also has some desirable mathematical properties, such as time additivity, which means that the log return over multiple periods is simply the sum of the log returns over each individual period.

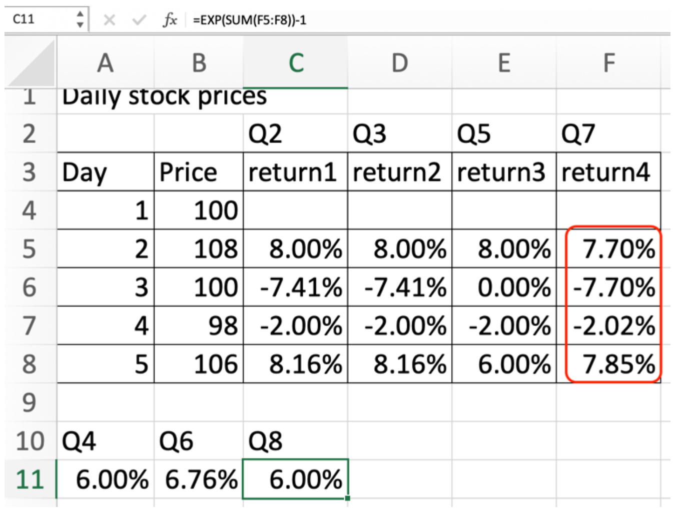
Also, note that we need to ensure that the denominator (S_t in this case) is not zero to avoid division by zero error. This can be handled by adding a small constant to the denominator when implementing the calculation in programs.

Q8: Calculate the terminal return using the log returns. Is it equal to Q4?

Answers: The terminal return using log returns can be calculated by summing all the single-period log returns, then exponentiating the result to reverse the log operation, and finally subtracting one to convert back to the simple return format. This is because log returns are time additive, meaning that the total log return over a given period is simply the sum of the log returns over the sub-periods.

In other words, if you have calculated log returns over several periods (say daily), you can get the total (terminal) log return over these periods simply by summing up all these daily log returns. This property simplifies the calculation of terminal returns over multiple periods, making it very convenient, especially for large datasets.

The result shows that it is equal to the one obtained in Q4. See Figure 5-8 for an illustration.



C11 =EXP(SUM(F5:F8))-1

	A	B	C	D	E	F
1	Daily stock prices					
2			Q2	Q3	Q5	Q7
3	Day	Price	return1	return2	return3	return4
4	1	100				
5	2	108	8.00%	8.00%	8.00%	7.70%
6	3	100	-7.41%	-7.41%	0.00%	-7.70%
7	4	98	-2.00%	-2.00%	-2.00%	-2.02%
8	5	106	8.16%	8.16%	6.00%	7.85%
9						
10	Q4	Q6	Q8			
11	6.00%	6.76%	6.00%			

Figure 5-8 Calculating the terminal return using the log returns.

Q9: Discuss the advantages of using log returns.

Answer: As mentioned, the use of logarithmic returns, or "log returns," has several advantages, as detailed in the following.

- Ease of calculation and analysis: Log returns simplify mathematical calculations and statistical analyses. This simplification is particularly noticeable when dealing with compounded returns over multiple periods. Because logarithms convert multiplication and division operations into addition and subtraction, the compounded return (or "total return") over multiple periods can be calculated as the simple sum of the log returns over those periods.
- Symmetry: Log returns also exhibit a desirable symmetry property. If a price doubles and then halves, or halves and then doubles, the total log return over the two periods is zero, reflecting the fact that the price is unchanged over the two periods. This symmetry property, which is not possessed by simple returns, often simplifies analyses and improves the interpretability of results.
- Suppose a stock price S_t changes to S_{t+1} and then changes back to S_t, the resulting log returns will be symmetric around zero. For example, when the stock price changes from 100 on day 1 to 108 on day 2 and then back to 100 on day 3, the resulting log returns are 7.7% on day 2 and -7.7% on day 3. A simple mathematical analysis would immediately make sense of this:

$$\log\frac{S_{t+1}}{S_t} = -\log\left(\frac{S_{t+1}}{S_t}\right)^{-1} = -\log\frac{S_t}{S_{t+1}}$$

- Normality: In addition, financial models often assume that returns are normally distributed. However, it's been observed that simple returns have skewness and excess kurtosis, implying that they deviate from normality. On the other hand, log returns tend to have properties closer to normality which makes them a better fit for these financial models.
- Continuously Compounded Returns: Log returns also represent continuously compounded returns. This property makes log returns the preferred choice in certain financial applications, especially those involving options and other derivatives, where continuous compounding is commonly used.

In summary, using log returns simplifies mathematical computations and statistical analyses, enables symmetry and normality, and represents continuously compounded returns. These properties make log returns highly valuable in financial analysis and modeling.

Let us look at a concrete example to understand the calculations using log returns.

Analyzing stock prices using log returns

We first download Google's stock price data for the first few days of 2023, as shown in Listing 5-1.

Listing 5-1. Downloading Google's stock price

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
symbol = 'GOOG'
df = yf.download(symbol, start="2023-01-01", end="2023-01-08")
>>> df
           Open       High       Low        Close Adj   Close
     Volume
Date
2023-01-03 89.830002  91.550003  89.019997  89.699997  89.699997
     20738500
2023-01-04 91.010002  91.239998  87.800003  88.709999  88.709999
     27046500
2023-01-05 88.070000  88.209999  86.559998  86.769997  86.769997
     23136100
2023-01-06 87.360001  88.470001  85.570000  88.160004  88.160004
     26612600
```

We can use the `pct_change()` method to calculate the single-period percentage returns, as shown in the Listing 5-2.

Listing 5-2. Calculating the single-period percentage returns

```
# single-period percentage returns
returns = df.Close.pct_change()
>>> returns
Date
2023-01-03 00:00:00-05:00         NaN
2023-01-04 00:00:00-05:00   -0.011037
2023-01-05 00:00:00-05:00   -0.021869
2023-01-06 00:00:00-05:00    0.016019
Name: Close, dtype: float64
```

Here, the first-period return is `NaN` as there is no prior stock price available.

Let us calculate the terminal return using the original approach by taking the first and last closing prices as the inputs (based on the definition given earlier), as shown in Listing 5-3.

Listing 5-3. Calculating the terminal return using the original approach by definition

```
# terminal return
terminal_return = df.Close[-1]/df.Close[0] - 1
>>> terminal_return
-0.01716826464354737
```

We can also calculate the same value by compounding the (1+R) returns based on the `.cumprod()` function, as shown in Listing 5-4.

Listing 5-4. Calculating the same cumulative terminal return by compounding 1+R formatted returns

```
# cumulative returns
cum_returns = (1+returns).cumprod() - 1
>>> cum_returns
Date
2023-01-03 00:00:00-05:00         NaN
2023-01-04 00:00:00-05:00   -0.011037
2023-01-05 00:00:00-05:00   -0.032664
2023-01-06 00:00:00-05:00   -0.017168
Name: Close, dtype: float64
```

The equality operator on both terminal returns evaluates to `True`.

```
# check equality on terminal return
>>> cum_returns.values[-1] == terminal_return
True
```

Now we calculate the same using log returns, starting by obtaining the single-period log returns in Listing 5-5.

Listing 5-5. Calculating the log returns

```
# log returns (1+R format)
log_returns = np.log(1+returns)
>>> log_returns
Date
2023-01-03 00:00:00-05:00          NaN
2023-01-04 00:00:00-05:00    -0.011098
2023-01-05 00:00:00-05:00    -0.022112
2023-01-06 00:00:00-05:00     0.015892
Name: Close, dtype: float64
```

We can add all log returns from previous periods together to get the cumulative log returns, convert to back to the original scale via exponentiation, and lastly, offset by 1 to convert from 1+R to the simple return format, as shown in Listing 5-6.

Listing 5-6. Calculating the cumulative returns using log returns

```
# get cumulative returns using log returns
cum_return2 = np.exp(log_returns.cumsum()) - 1
>>> cum_return2
Date
2023-01-03 00:00:00-05:00          NaN
2023-01-04 00:00:00-05:00    -0.011037
2023-01-05 00:00:00-05:00    -0.032664
2023-01-06 00:00:00-05:00    -0.017168
Name: Close, dtype: float64
```

Again, we verify the value of the last entry and verify that it is the same as the previous terminal return.

```
# check equality on terminal return
>>> cum_return2.values[-1] == terminal_return
True
```

The next section introduces the trend-following strategy.

Introducing trend trading

Trend trading, also known as trend following, is a strategy that attempts to harness the momentum of an existing trend in a financial market. It operates on the premise that securities tend to move in a relatively sustained direction over time, either upwards (bullish) or downwards (bearish). It is a proactive trading strategy that seeks to capitalize on the sustained directional momentum of an asset's price.

The fundamental principle behind trend trading is that a market's momentum, or the rate of acceleration of the asset's price, often continues in one direction for a period of time. This is where the two key concepts, trend and momentum, come into play. The trend represents the direction in which an asset's price is moving, while momentum indicates the strength or speed of this movement over a certain period. It refers to the capacity for the asset's price trend to sustain itself going forward. A strong momentum can continue in an upward or downward trend, which can be confirmed by a set of technical indicators.

Trend traders leverage technical analysis tools to identify potential buying and selling opportunities. They carefully analyze price charts and use various technical indicators, such as moving averages, MACD (Moving Average Convergence Divergence), and the relative strength index (RSI), among others, to identify and confirm an asset's trend direction and momentum. These technical indicators provide signals that help traders to make educated decisions about when to enter and exit trades.

In an uptrend, a trend trader will enter a long position, meaning they buy the asset with the expectation that its price will continue to rise. Conversely, in a downtrend, a trend trader will enter a short position, meaning they sell the asset (or sell short) with the expectation that its price will continue to fall. The trend-following strategy aims to take advantage of these significant movements in price and to profit from both rising and falling markets based on the forward-looking uptrends with new highs or anticipated downtrends with new lows.

Let us start with the technical indicators which are used to generate trading signals.

Understanding technical indicators

Technical indicators are mathematical calculations based on historical price (high, low, open, close, etc.) or volume, and can be used to determine entry and exit points for trades. They are integral to many trading strategies and systems, providing key insights into market behavior. They can be considered as additional features derived from the raw asset data, a practice of feature engineering in machine learning. This makes technical indicators highly security-dependent: what can be a good technical indicator for a particular security might not hold the case for the other. Selecting the right features makes all the difference.

Note that these technical indicators appear as additional features for each observation in the dataset. This means that more columns are added to the price-volume table we worked with earlier, with each column representing a separate technical indicator for the specific asset and time.

When looking at the raw price data, overlaying a set of technical indicators would help clarify the market analysis for traders. For example, technical indicators help confirm if the market is following a trend or in a range-bound situation, oscillating within a price range.

Technical indicators are integral to many trading strategies and systems, providing key insights into market behavior. As you've described, they are tools derived from mathematical calculations on historical price and volume data, designed to predict future price trends or patterns.

Some of the most commonly used technical indicators include:

- Moving Averages (MA): Moving averages smooth out price data by creating a constantly updated average price. The two most common types are the Simple Moving Average (SMA) and the Exponential Moving Average (EMA). They can help identify whether a security is in an uptrend or downtrend. More on this later.
- Relative Strength Index (RSI): The RSI measures the speed and change of price movements, typically on a scale of 0 to 100. A high RSI (generally above 70) may indicate that the asset is overbought and due for a price correction, while a low RSI (generally below 30) could suggest that the asset is oversold and might rebound.
- Moving Average Convergence Divergence (MACD): This indicator is a trend-following momentum indicator that shows the relationship between two moving averages of a security's price. The MACD is calculated by subtracting the 26-day EMA from the 12-day EMA.
- Bollinger Bands: These bands are plotted two standard deviations away from a simple moving average. They help identify whether an asset is overbought or oversold and can signal the end of a trend.
- Volume-based Indicators: These include indicators such as the On-Balance-Volume (OBV), which uses volume flow to predict changes in stock price.

Each of these indicators provides a unique perspective on potential market movements. A combination of these indicators is often used to create a robust trading strategy.

Also, note that these indicators don't predict future prices with absolute certainty. Instead, they help traders identify potential trading opportunities based on statistical probabilities. Each indicator works best under specific market conditions and may not be universally applicable across different asset classes, markets, and trading horizons.

The following section provides more introduction to moving averages.

Introducing moving averages

Moving average, also called rolling average, is the mean or average of the specified data field (e.g. daily closing price) for a given set of *consecutive* periods. As new data becomes available, the mean of the data is computed by dropping the oldest value and adding the latest one. It is rolling along with the data, hence the name “Moving Average”. It provides a way of smoothing out the price data of a financial asset to identify trends more clearly.

When calculating moving averages of stock prices, it works similarly to moving a fixed-sized window along the time horizon, where each window reports a single number as the average of all price points within the window. And when that window does not have full price points for the initial periods, an NA value is often reported.

When working with time series data such as daily stock price, the averaging effect can also be considered as smoothening the time series, reducing short-term fluctuations and temporary variations in the data.

There are different types of moving averages, with the simple moving average and the exponential moving average being the most popular ones. The simple moving average is straightforward to calculate; we simply take the average of all the price points in the current fixed-size window, assuming an equal weightage for all the price points in this window.

The exponential moving average, or exponentially weighted moving average (EWMA), decreases the weightage for older price points. It's more complex to calculate than the SMA, as it involves a smoothing factor that needs to be computed. But the basic idea is the same: it's an average of the closing prices over a certain period.

The choice between using a simple or exponential moving average depends on the trader's preference and the specific trading strategy. In general, EMAs react more quickly to recent price changes than SMAs, making them more preferred by short-term traders or those trading volatile markets.

Moving averages can be used to identify support and resistance levels. The support level is typically a price level or zone that a stock or a market has had difficulty falling below over a specific period. The resistance level is the opposite of the support level. It's a price level or zone that a stock or a market has trouble moving above. Prices often bounce off these levels, making them useful for identifying potential trade entry and exit points.

In addition, when two moving averages (e.g., 50-day and 200-day) cross each other, it may signal a change in trend. A bullish signal is given when the shorter MA crosses above the longer MA, and a bearish signal is given when the shorter MA crosses below the longer MA. These crossover points become potential trading signals.

The following section focuses more on the simple moving averages.

Delving into simple moving averages

The simple moving average SMA_t at time t is defined as follows:

$$\mathrm{SMA}_t = \frac{S_{t-(M-1)} + \cdots + S_{t-1} + S_t}{M}$$

In other words, to calculate SMA_t, we would take M historical price points, including the current period, and then take the average of these M price points. Essentially, it involves adding up the prices of the security for the last M periods (days, hours, etc.), and then dividing by M. This provides a single output point, the SMA at time t. As new price data becomes available, the oldest data point is dropped, and the newest data point is included in the calculation. This "rolling" or "moving" calculation continues as new price data is added.

The SMA is often used in trend analysis as it smoothens out short-term fluctuations and provides a clearer picture of the overall trend. It is the unweighted mean of the previous M price points. Here, the choice of M (the number of periods) is crucial because it affects the sensitivity and reliability of the SMA. A smaller M will be more responsive to price changes but may also yield more false signals. A larger M will provide a slower, more reliable SMA, but it might be slower in signaling changes in trends.

Let us look at how to calculate SMA. We first download Apple's stock price data for 2022, as shown in Listing 5-6.

Listing 5-6. Downloading Apple's stock price data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
symbol = 'AAPL'
df = yf.download(symbol, start="2022-01-01", end="2023-01-01")
df.index = pd.to_datetime(df.index)
>>> df.head()
           Open       High       Low        Close Adj   Close
      Volume
Date
2022-01-03 177.830002 182.880005 177.710007 182.009995 180.434296
      104487900
2022-01-04 182.630005 182.940002 179.119995 179.699997 178.144302
      99310400
2022-01-05 179.610001 180.169998 174.639999 174.919998 173.405685
      94537600
2022-01-06 172.699997 175.300003 171.639999 172.000000 170.510956
      96904000
2022-01-07 172.889999 174.139999 171.029999 172.169998 170.679489
      86709100
```

Note that we have an index named `Date` which now assumes a `datetime` format to facilitate plotting.

Listing 5-7 generates a plot on the daily adjusted closing price. We will later overlay its SMA on the same plot.

Listing 5-7. Plotting the daily adjusted closing price

```
# plot the adj closing price
plt.figure(figsize=(15, 7))
df['Adj Close'].plot()
# set labels and sizes of the title and axis
plt.title('Daily adjusted closing price of Apple', fontsize=16)
plt.xlabel('Time', fontsize=15)
plt.ylabel('Price ($)', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(['Close'], prop={'size': 15})
# show the plot
>>> plt.show()
```

Running the above commands generates Figure 5-9, suggesting a download trend overall.

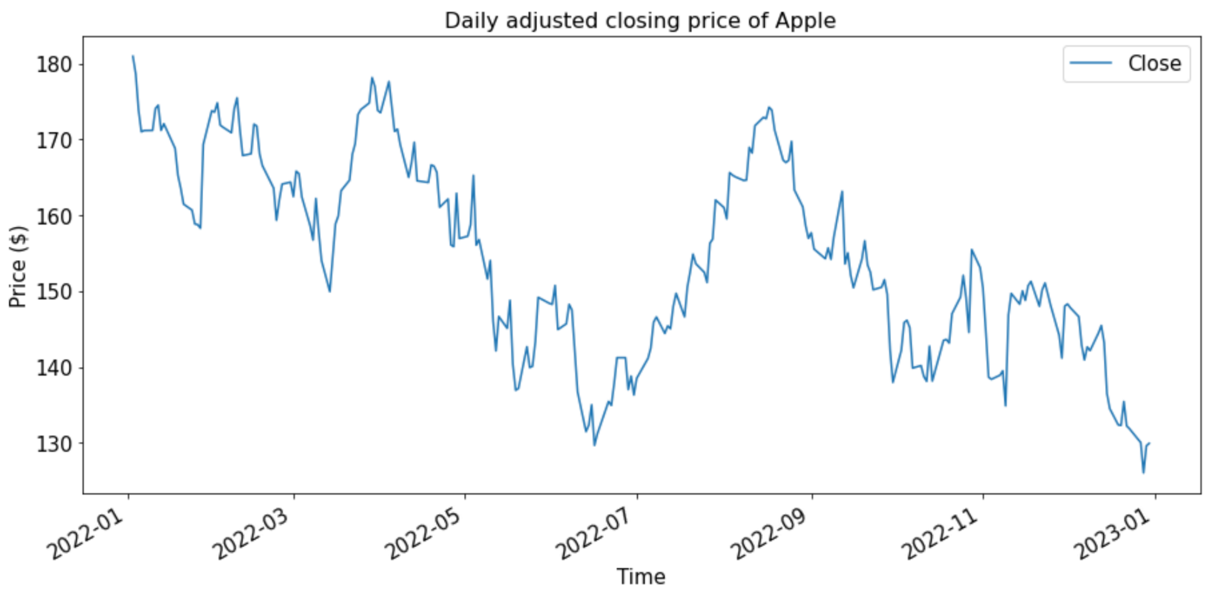


Figure 5-9 Visualizing the daily closing price of Apple in 2022.

Now we create an SMA series with a window size of 3. We can create the rolling window using the `rolling()` method for a Pandas series, followed by the `mean()` method to extract the average value from the window (a collection of price points). Listing 5-8 creates a new SMA column called `SMA-3` and subsets to keep only two columns: the adjusted closing price and the SMA column.

Listing 5-8. Creating simple moving averages

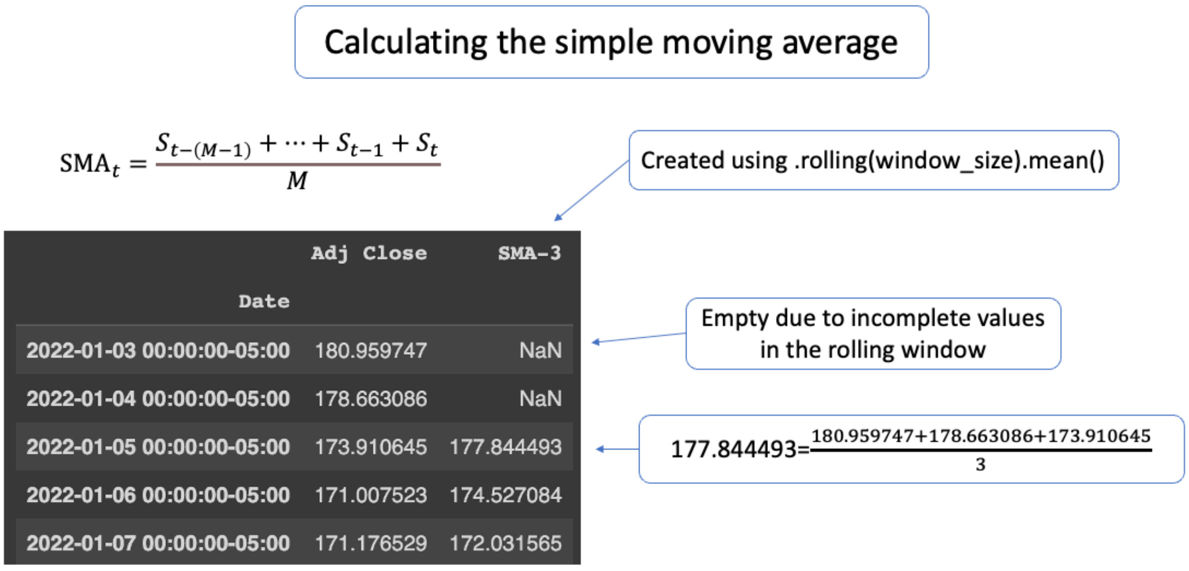
```
window = 3
SMA1 = "SMA-"+str(window)
df[SMA1] = df['Adj Close'].rolling(window).mean()
colnames = ["Adj Close",SMA1]
df2 = df[colnames]
>>> df2.head()
           Adj Close  SMA-3
Date
2022-01-03 180.434296 NaN
2022-01-04 178.144302 NaN
2022-01-05 173.405685 177.328094
2022-01-06 170.510956 174.020315
2022-01-07 170.679489 171.532043
```

Let us pause for a moment and look at how this column is generated. We see that the first two rows in the SMA column are missing. This makes sense as both of them are unable to get a full 3-period moving window to calculate the average. In other words, we cannot calculate the average when there is an empty value in the window unless additional treatment is applied here, such as ignoring the empty value while calculating the average.

We note that the third entry of the SMA column is 177.844493. Let us verify through manual calculation. The following command takes the first 3 entries of the adjusting closing price column and calculates the average, which reports the same value.

```
>>> np.mean(df['Adj Close'][:3])
177.84449259440103
```

which verifies the calculation. Figure 5-10 summarizes the process of calculating SMA in our running example.


Calculating the simple moving average
$\mathrm{SMA}_t = \frac{S_{t-(M-1)} + \cdots + S_{t-1} + S_t}{M}$
Created using .rolling(window_size).mean()
Empty due to incomplete values in the rolling window
$177.844493 = \frac{180.959747+178.663086+173.910645}{3}$

	Adj Close	SMA-3
Date		
2022-01-03 00:00:00-05:00	180.959747	NaN
2022-01-04 00:00:00-05:00	178.663086	NaN
2022-01-05 00:00:00-05:00	173.910645	177.844493
2022-01-06 00:00:00-05:00	171.007523	174.527084
2022-01-07 00:00:00-05:00	171.176529	172.031565

Figure 5-10 Illustrating the process of calculating simple moving averages.

Note that we can configure the `min_periods` argument in the `rolling()` function to control the behavior at the initial windows with incomplete data. For example, by setting `min_periods=1`, the previous codes will report the average value based on the *available* data in the window. See the following code snippet for a comparison.

```
df['New_SMA'] = df['Adj Close'].rolling(window,
min_periods=1).mean()
>>> df[colnames + ['New_SMA']].head()
           Adj Close  SMA-3      New_SMA
Date
2022-01-03 180.434296 NaN   180.434296
2022-01-04 178.144302 NaN   179.289299
2022-01-05 173.405685 177.328094 177.328094
2022-01-06 170.510956 174.020315 174.020315
2022-01-07 170.679489 171.532043 171.532043
```

Note that the only difference is in the first two entries, where we have an incomplete set of values in the rolling window.

Next, we plot the 3-period SMA alongside the original daily adjusted close price series, as shown in Listing 5-9.

Listing 5-9. Plotting the closing price and its SMA

```
# colors for the line plot
colors = ['blue', 'red']
# line plot for original price and SMA
df2.plot(color=colors, linewidth=3, figsize=(12,6))
# modify ticks size
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.legend(labels = colnames, fontsize=13)
# title and labels
plt.title('Daily adjusted closing price and its SWA', fontsize=20)
plt.xlabel('Date', fontsize=16)
plt.ylabel('Price', fontsize=16)
```

Running these commands generates Figure 5-11. Note that the 3-period SMA curve in red looks less volatile than the original price series in blue. Also, the 3-period SMA curve starts from the third entry.

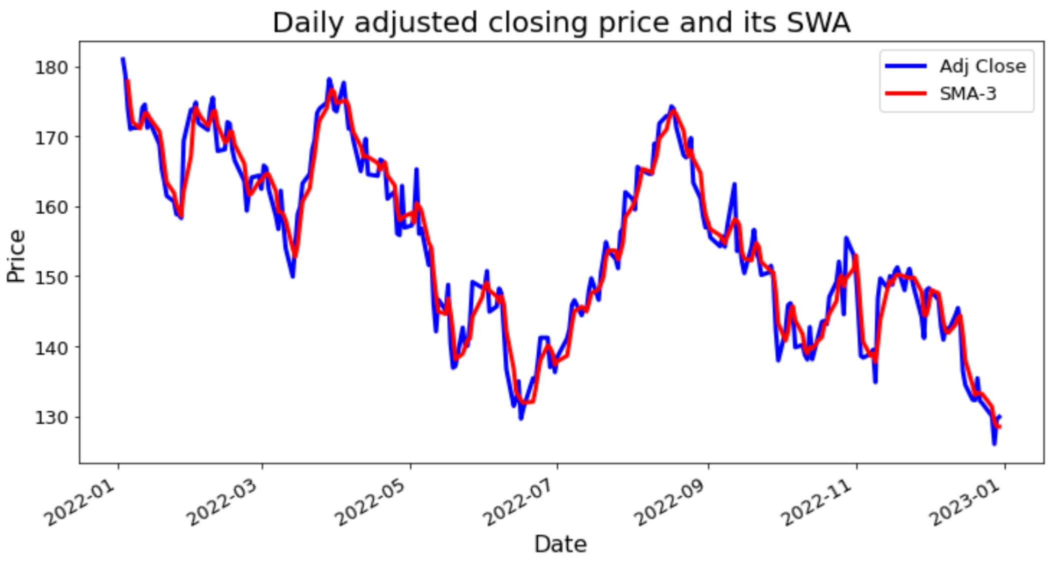


Figure 5-11 Visualizing the original price and 3-period SMA.

Now let us add another SMA with a longer period. In Listing 5-10, we add a 20-period SMA as an additional column to `df2`.

Listing 5-10. Creating 20-period SMA

```
window = 20
SMA2 = "SMA-"+str(window)
df2["SMA-"+SMA2] = df2['Adj Close'].rolling(window).mean()
```

```
colnames = ["Adj Close",SMA1,SMA2]
```

Next, we overlay the 20-period SMA on the previous graph, as shown in Listing 5-11.

Listing 5-11. Plotting the closing price and two SMAs

```
# colors for the line plot
colors = ['blue', 'red', 'green']
# line plot for original price and SMA
df2.plot(color=colors, linewidth=3, figsize=(12,6))
# modify ticks size
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.legend(labels = colnames, fontsize=13)
# title and labels
plt.title('Daily adjusted closing price and its SWA', fontsize=20)
plt.xlabel('Date', fontsize=16)
plt.ylabel('Price', fontsize=16)
```

Running these commands generates Figure 5-12, which shows that the 20-period SMA is smoother than the 3-period SMA due to a larger window size.

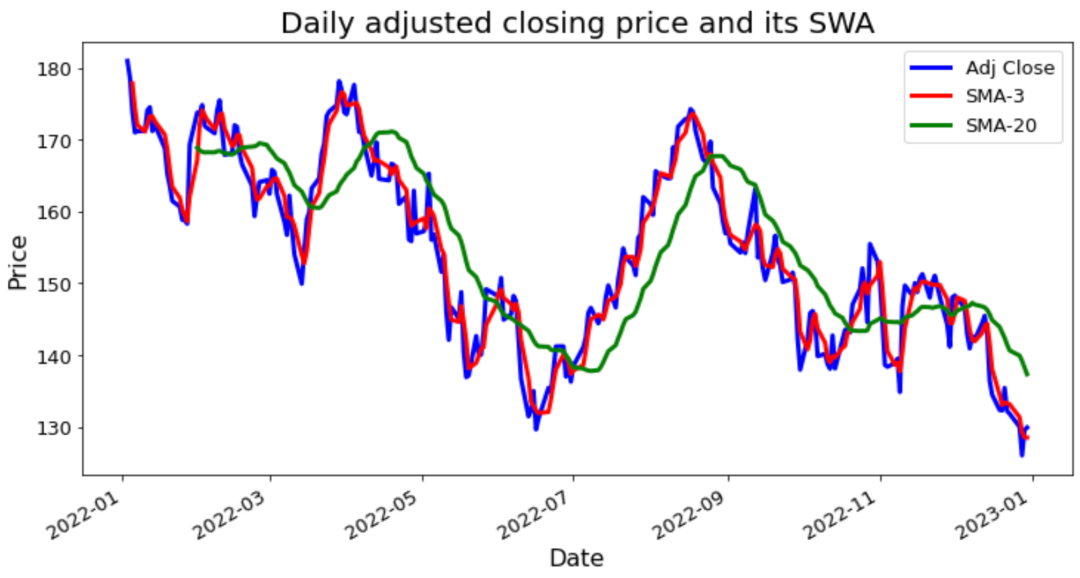


Figure 5-12 Visualizing the daily prices together with 3-period and 20-period SMAs.

The next section focuses on the exponential moving averages (EWA).

Delving into exponential moving averages

The Exponential Moving Average (EMA), also known as an Exponentially Weighted Moving Average (EWMA), is another type of moving average that places a higher weight and significance on the most recent data points. This is a key difference compared to the simple moving average, which gives equal weight to all data points within the period.

The exponential moving average (EMA) is a widely used method to reduce the noise in the data and identify long-term trends. Each EMA entry is a weighted combination of historical prices and the current price. The weight of each price point decreases progressively over time, giving greater weight to recent data points. It is calculated using the following formula:

$$\mathrm{EWMA}_t = \begin{cases} S_0, & t = 0 \\ \alpha S_t + (1-\alpha)\mathrm{EWMA}_{t-1}, & t > 0 \end{cases}$$

Where α is the smoothing factor which ranges between 0 and 1. The smoothing factor α determines the weight given to the most recent price relative to the existing EMA. A higher α emphasizes recent prices more strongly.

As for the first EWMA value at time $t = 0$, a default choice is to set $\mathrm{EWMA}_0 = S_0$. Therefore, EMA assumes that recent data is more relevant than old data. Such an assumption has its merit since EMA can react faster to changes and is thus more sensitive to recent movements as compared to the simple moving average. This also means there is no window size to be specified by the function since all historical data points are in use.

It's important to note that while EMA provides more accurate and timely signals than SMA, it might also produce more false signals as it's more responsive to short-term price fluctuations.

The EMA can be calculated by calling the `ewm()` method from a Pandas series object, followed by extracting the average value via `mean()`. We can set the `alpha` argument in `ewm()` to directly control the importance of the current observation compared with historical ones. See Listing 5-12 for an illustration, where we set $\alpha = 0.1$ to give more weightage to historical prices.

Listing 5-12. Creating EMA series

```
alpha = 0.1
df2['EWM_'+str(alpha)] = df2['Adj Close'].ewm(alpha=alpha,
adjust=False).mean()
df2.head()
           Adj Close  SMA-3 SMA-20 EWM_0.1
Date
2022-01-03 180.434296 NaN   NaN   180.434296
2022-01-04 178.144302 NaN   NaN   180.205296
2022-01-05 173.405685 177.328094 NaN   179.525335
2022-01-06 170.510956 174.020315 NaN   178.623897
2022-01-07 170.679489 171.532043 NaN   177.829456
```

We observe that there is no missing value in the EMA series. Indeed, the first entry will simply be the original price itself due to the design of the EMA weighting scheme.

As usual, let us verify the calculations to ensure our understanding is on the right track. The following code snippet manually calculates the second EMA value, which is the same as the one obtained using `ewm()` function.

```
alpha=0.1
>>> alpha*df2['Adj Close'][1] + (1-alpha)*df2['Adj Close'][0]
180.73006591796877
```

Let us continue to create another EMA series with $\alpha = 0.5$. In other words, we assign an equal weightage to the current observation and historical ones.

```
alpha = 0.5
df2['EWM_'+str(alpha)]= df2['Adj Close'].ewm(alpha=alpha,
adjust=False).mean()
df2.head()
           Adj Close  SMA-3 SMA-20     EWM_0.1    EWM_0.5
Date
2022-01-03 180.434296 NaN   NaN   180.434296 180.434296
2022-01-04 178.144302 NaN   NaN   180.205296 179.289299
2022-01-05 173.405685 177.328094 NaN   179.525335 176.347492
2022-01-06 170.510956 174.020315 NaN   178.623897 173.429224
2022-01-07 170.679489 171.532043 NaN   177.829456 172.054357
```

Let us put all these moving averages in a single chart. Here, the `plot()` function treats all four columns as four separate series to be plotted against the index column, as shown in Listing 5-13.

Listing 5-13. Plotting all moving averages together

```
df2.plot(linewidth=3, figsize=(12,6))
plt.title('Daily adjusted closing price with SWA and EWM',
fontsize=20)
plt.xlabel('Date', fontsize=16)
plt.ylabel('Price', fontsize=16)
```

Running these commands generates Figure 5-13. We note that `EWM_0.1` (red line) is close to `SMA-20` (green line), both of which give more weightage to historical observations. The same the true for the other two moving averages. For EMA, a small weighting factor α results in a high degree of smoothing, while a larger value leads to a quicker response to recent changes.

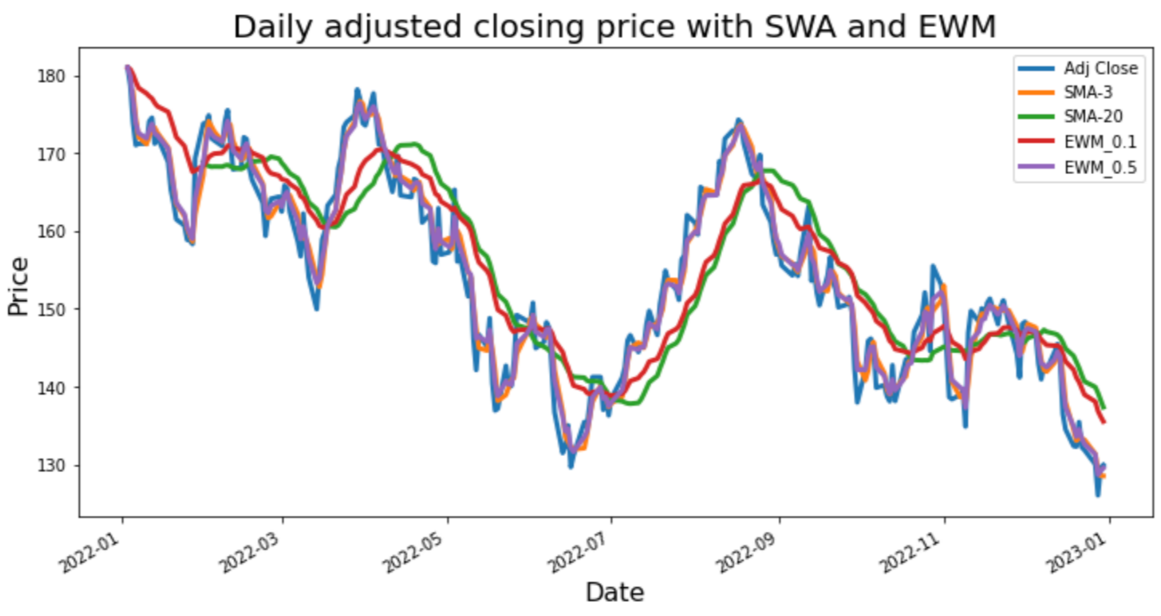


Figure 5-13 Visualizing the daily closing prices with both SMA and EMA of different configurations.

Having looked at how to compute these moving averages, the next section shows how to use them as technical indicators to develop a trend-following strategy.

Implementing the trend-following strategy

The trend-following strategy that relies on moving averages works like this. There will be two moving averages: a short-term moving average and a long-term moving average. When the short-term moving average cross above the long-term moving average, it signals a buy action, and the trend trader enters a long position on the asset. When the short-term moving average cross below the long-term moving average, it signals a sell action, and the trend trader enters a short position on the asset. Thus the strategy is based on the intersection of two moving averages: one short-term (quick) and one long-term (slow).

Note that this framework also applies to the case when there is only one moving average series. In this case, the trend trader would buy the asset when the current price is above the moving average and sell it if the current price is below the moving average. The key justification for such trading action is, when the price is above a moving average, an uptrend may be present, and vice versa. The *crossover* between two lines generates the trading signal.

Other momentum-related technical indicators, such as the RSI and MACD, may also be used to signal entries or exits.

In the following section, we will implement a trend-following trading strategy using the long-term and short-term moving averages. Using this strategy, we are essentially searching for the trading signal at each time point. That is, we want to decide if we would buy, sell, or hold an asset at each time step. The signal is generated by a crossover between two moving averages. We assume no transaction cost will be incurred when performing a trading action, and the market is liquid (sufficient Apple stock in the market) and complete (no arbitrage opportunities).

Let us recall the main DataFrame we will work with. The following command prints out the summary information using the `info()` function.

```
>>> df2.info()
 <class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 251 entries, 2022-01-03 00:00:00-05:00 to 2022-12-30
00:00:00-05:00
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Adj Close  251 non-null    float64
 1   SMA-3      249 non-null    float64
 2   SMA-20     232 non-null    float64
 3   EWM_0.1    251 non-null    float64
 4   EWM_0.5    251 non-null    float64
dtypes: float64(5)
memory usage: 19.9 KB
```

Now we will use `SMA-3` and `SMA-20` as the respective short-term and long-term moving averages, whose crossover will generate a trading signal. We leave it as an exercise to try both SMA with different window sizes and EMA with different weighting schemes.

Note that we can only use the information up to yesterday to make a trading decision for tomorrow. We cannot use today's information since the closing price is not yet available in the middle of the day. To enforce this requirement, we can shift the moving averages one day into the future, as shown in the following code snippet. This essentially says that the moving average for today is derived from historical information up to yesterday.

```
# Shift to the future by one day so that everyday uses the
information up to
# yesterday to make a trading decision for tmr
df2['SMA-3'] = df2['SMA-3'].shift(1)
df2['SMA-20'] = df2['SMA-20'].shift(1)
```

Now let us implement the trading rule: buy if `SMA-3 > SMA-20`, and sell if `SMA-3 < SMA-20`. Such an `if-else` condition can be created using the `np.where()` function, as shown in Listing 5-14.

Listing 5-14. Creating and buy and sell signals

```
# identify buy signal
df2['signal'] = np.where(df2['SMA-3'] > df2['SMA-20'], 1, 0)
# identify sell signal
df2['signal'] = np.where(df2['SMA-3'] < df2['SMA-20'], -1,
df2['signal'])
df2.dropna(inplace=True)
```

Here, a normal trading day would assume a value of either 1 or -1 in the signal column. When there is a missing value or other special cases, we set it to 0. We also use the `dropna()` function to ensure that the DataFrame is of good quality by dropping rows with any NA/missing value in it.

We can check the frequency distribution of the signal column as follows.

```
>>> df2['signal'].value_counts()

-1    135
 1     96
Name: signal, dtype: int64
```

The result shows that there are more declining days than inclining days, which confirms the downward trending price series shown earlier.

Next, we introduce a baseline strategy called *buy-and-hold*, which simply means we hold one share of Apple stock until the end of the whole period. Also, we will use the log return instead of the raw return to facilitate the calculations. Therefore, instead of taking the division between consecutive stock prices to get $\frac{S_{t+1}}{S_t}$, we now take the difference $\log S_{t+1} - \log S_t$ to get $\log \frac{S_{t+1}}{S_t}$, which can then be exponentiated to convert to back $\frac{S_{t+1}}{S_t}$.

The following code snippet calculates the instantaneous logarithmic single-period return, where we first take the logarithm of the adjusted closing prices, and then call the `diff()` function to obtain the differences between consecutive pairs of prices.

```
df2['log_return_buy_n_hold'] = np.log(df2['Adj Close']).diff()
```

Now comes the calculation of the single-period return for the trend-following strategy. Recall the `signal` column we created earlier. This column represents whether we go long (valued 1) or short (value -1) in a position for every single period. This also shows that the logarithmic return $\log \frac{S_{t+1}}{S_t}$ is positive if $S_{t+1} > S_t$ and negative if $S_{t+1} < S_t$. This creates the following four scenarios when the asset moves from S_t to S_{t+1}:

- When we long an asset and its logarithmic return is positive, the trend-following strategy reports a positive return, i.e., $1 * \log \frac{S_{t+1}}{S_t}$;
- When we long an asset and its logarithmic return is negative, the trend-following strategy reports a negative return, i.e., $1 * \log \frac{S_{t+1}}{S_t}$;
- When we short an asset and its logarithmic return is positive, the trend-following strategy reports a negative return, i.e., $-1 * \log \frac{S_{t+1}}{S_t}$;
- When we short an asset and its logarithmic return is negative, the trend-following strategy reports a positive return, i.e., $-1 * \log \frac{S_{t+1}}{S_t}$;

Summarizing these four scenarios, we can obtain the single-period logarithmic return for the trend-following strategy by multiplying `signal` with the `log_return_buy_n_hold` (the single-period logarithmic return based on the buy-and-hold strategy), as shown in Listing 5-15.

Listing 5-15. Calculating the log return of trend following strategy

```
df2['log_return_trend_follow'] = df2['signal'] *
df2['log_return_buy_n_hold']
```

Compared with the buy-and-hold strategy, the key difference is the additional shorting actions generated by the trend-following strategy. That is, when the stock price drops, the buy-and-hold strategy will register a loss, while the trend-following strategy will make a profit *if* the trading signal is to go short. Creating a good trading signal thus makes all the difference.

Next, we create explicit trading actions. The `signal` column tells us whether we should go long or short in the given asset under the trend-following strategy. However, this does not mean we need to make a trade at every period. If the `signal` remains the same for two consecutive periods, we simply hold on to the position and remain seated. In other words, there is no trading action for this specific trading day. This applies in the case of two consecutive 1s or -1s in the `signal` column.

On the other hand, we will make an action when there is a sign switch in the trading signal, changing from 1 to -1 or from -1 to 1. The former means changing from longing a unit of stock to shorting it, while the latter means the reverse.

To create the trading actions, we can use the `diff()` method again on the `signal` column, as shown in the following.

```
df2['action'] = df2.signal.diff()
```

We can produce a frequency count of different trading actions using the `value_counts()` function.

```
>>> df2['action'].value_counts()

0.0    216
 2.0      7
-2.0      7
Name: action, dtype: int64
```

The result shows that the majority of the trading days do not require action. For the 14 days with a trading action, 7 days change the position from short to long, and another 7 change from long to short.

We can visualize these trading actions as triangles on the graph with stock prices and SMAs. In Listing 5-16, we indicate a buy action via the green triangle facing upwards when the short-term SMA crosses above the long-term SMA. On the other hand, we use a red triangle facing downward to indicate a sell action when the short-term SMA crosses below the long-term SMA.

Listing 5-16. Visualizing trading actions

```
plt.rcParams['figure.figsize'] = 12, 6
plt.grid(True, alpha = .3)
plt.plot(df2['Adj Close'], label = 'Adj Close')
plt.plot(df2['SMA-3'], label = 'SMA-3')
plt.plot(df2['SMA-20'], label = 'SMA-20')
plt.plot(df2.loc[df2.action == 2].index, df2['SMA-3'][df2.action ==
2], '^',
         color = 'g', markersize = 12)
plt.plot(df2[df2.action == -2].index, df2['SMA-20'][df2.action == -
2], 'v',
         color = 'r', markersize = 12)
plt.legend(loc=1);
```

Running these commands generates Figure 5-14. Again, we denote the green triangles as acting from short to long and the red triangles as moving from long to short.

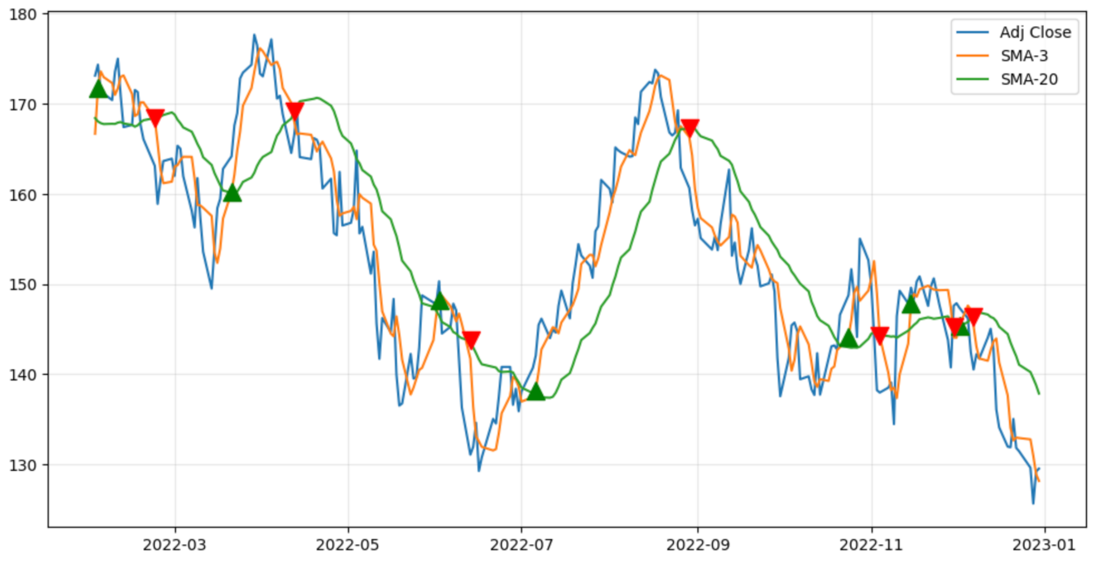


Figure 5-14 Visualizing the trading actions, including going from short to long (green triangles) and long to short (red triangles).

Let us analyze the cumulative returns of each period for both trading strategies. Specifically, we would like to obtain the final percentage return at the end of 2022 if we started with one unit of Apple stock at the beginning of 2022, comparing the two trading strategies.

Recall that we need to multiply the 1+R return at each period to carry out the compounding process in order to obtain the terminal return (after subtracting 1). We also know that the 1+R return is the same as the division between two consecutive prices, i.e., $1 + R_{t,t+1} = \frac{S_{t+1}}{S_t}$. Therefore, to calculate the terminal return, we first convert the returns from the logarithmic format to the usual percentage format using the `np.exp()` function, carry out the compounding by performing a cumulative product operation using the `cumprod()` method. This is achieved via Listing 5-17, where we leave out the last step of subtracting by 1 and report the 1+R return.

Listing 5-17. Visualizing cumulative returns

```
plt.plot(np.exp(df2['log_return_buy_n_hold']).cumprod(), label='Buy-
n-hold')
plt.plot(np.exp(df2['log_return_trend_follow']).cumprod(),
label='Trend following')
plt.legend(loc=2)
plt.title("Cumulative return of different trading strategies")
plt.grid(True, alpha=.3)
```

Running these commands generates Figure 5-15, which shows that the trend-following strategy clearly outperforms the buy-and-hold strategy. However, note that this is a simplified setting that does not take into account transaction cost and other market factors. More analyses and tests are needed to assess the performance of this trading strategy (also many others) in the real-world environment.

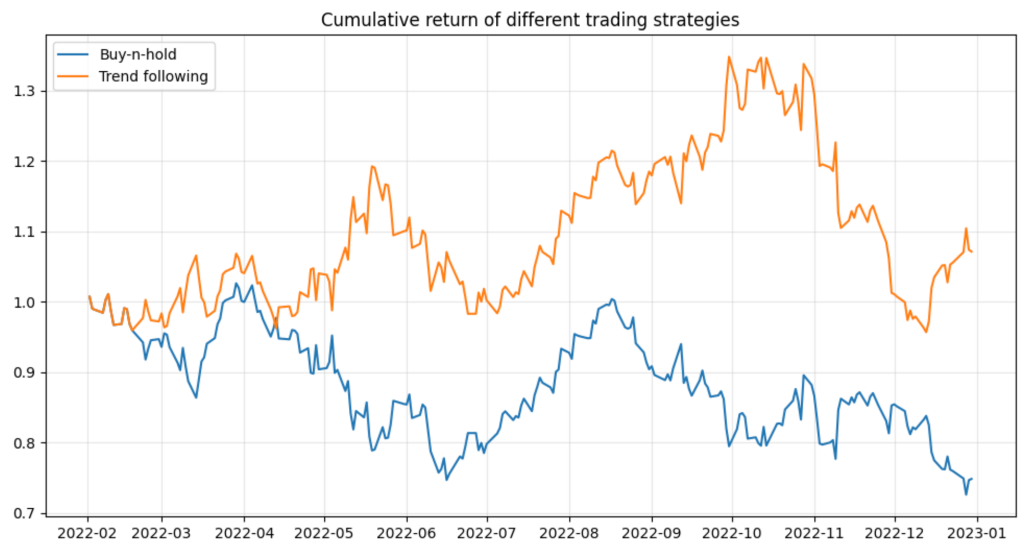


Figure 5-15 Comparing the cumulative return of buy-and-hold and trend-following strategies for one share of Apple's stock.

Lastly, we compare the terminal returns of both strategies.

```
# terminal return of buy-n-hold
>>> np.exp(df2['log_return_buy_n_hold']).cumprod()[-1] -1
-0.25156586984649587
# terminal return of trend following
>>> np.exp(df2['log_return_trend_follow']).cumprod()[-1] -1

0.0711944903093773
```

It turns out that sticking to the buy-and-hold strategy would lose by 25% while using the trend-following strategy generates a terminal return of 7%.

Summary

In this chapter, we covered the basics of the popular trend-following strategy and its implementation in Python. We started with an exercise on working with log returns, and then transitioned to different moving averages as commonly used technical indicators, including simple moving averages and exponential moving averages. Lastly, we discussed how to generate trading signals and calculate the performance metrics using this strategy, which will serve as a good baseline strategy as we delve into other candidates later on.

Exercises

- Explain why log returns are symmetric mathematically.
- How can we deal with a situation where the price point at a given day is missing when calculating its moving average?
- How does the value of the window size affect the smoothness of the SMA? What about the impact of α on the smoothness of EMA?
- Change the codes to obtain a moving median instead of a moving average. Discuss the difference between the median and the mean. How about maximum and minimum over the same rolling window?
- Switch to EMA to derive the trading signals and discuss the results.
- Show mathematically why the log returns are additive over time and explain the significance of this property in the context of asset returns.
- Suppose there are multiple missing price points in your data, how would you modify the moving average calculation to handle these gaps? What are the potential issues with your approach?
- Experiment with different window sizes for SMA and different values of α for EMA. Discuss how these parameters affect the sensitivity of the moving averages to price changes. How would you choose an optimal value for these parameters?