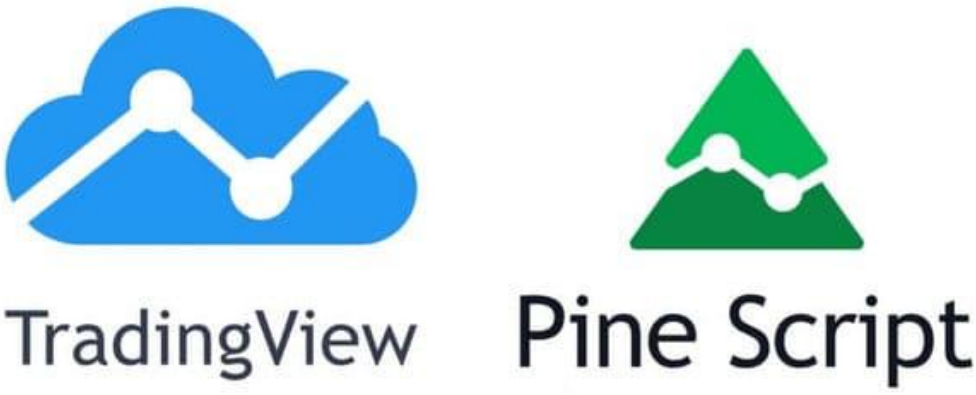
Pine Script (TradingView) – A Step-by-step Guide

https://algotrading101.com/learn/pine-script-tradingview-guide/


Jignesh Davda

This article has been updated for Pine Script V5.

Table of Contents



What is the Pine script?

Pine script is a programming language created by TradingView to backtest trading strategies and create custom indicators.

Pine script was designed to be lightweight, and in most cases, you can achieve your objectives with fewer lines of code compared to other programming languages.

It is not based on any particular language, but if you've used Python, you'll tend to pick it up quickly and notice similarities.

Pine script code can be created within Pine editor which is a part of TradingView's online charting platform.

Link: https://www.tradingview.com/pine-script-docs/en/v5/Introduction.html

Why should I use Pine script?

Built-in Data – This is a big one. Testing strategies or creating indicators in other languages involves sourcing your own data.

Not only does that mean you have to find a place to grab your data from, but you'll also then have to format it in a certain way and this whole process can be time-consuming.

TradingView has a plethora of data available at your fingertips, ready to access with as little as one line of code.

Easy to Learn – Pine script syntax is readable and simpler than other programming languages.

Also, you don't have to spend much time on error checking and handling as TradingView takes care of most of that for you.

Extensive user base and library – TradingView users have the option to publish their indicators and strategies to the TradingView library.

Having access to open-source code is a great way to learn from other programmers. Also, in some cases, someone else may have already written the code for what you're after.

» Quantopian has shut down. An alternative to consider is QuantConnect.

QuantConnect is a browser-based backtesting and algo trading platform.

Link: QuantConnect – A Complete Guide

Content Highlights:

- Create strategies based on alpha factors such as sentiment, crypto, corporate actions and macro data (data provided by QuantConnect).

- Backtest and trade a wide array of asset classes and industries ETFs (data provided by QuantConnect).
- License strategies to hedge funds (while you keep the IP) via QuantConnect's Alpha Stream.

Why shouldn't I use Pine script?

The main reason why you wouldn't want to use Pine script is that you're limited to the TradingView universe.

Here are some specific limitations –

Data – If TradingView does not offer the data you're after, you're out of luck.

Granted, TradingView has a very comprehensive database of data feeds. But if your strategy involves trading obscure markets, price data may not be available.

This extends outside of price data. Some strategies involve economic or statistical data. TradingView does offer some data (mainly Quandl data) in this category but it is limited at this time.

If you use alternative data in your strategy, it's probably easier to use another programming language that offers more flexibility.

External libraries – Pine script is not appropriate if you're looking to leverage external libraries to do things like Machine learning.

There are better alternatives if your strategy relies on using data science or other third-party libraries.

» Check out how we use TradingView to visually find pairs to trade.

Link: Pairs Trading – A Real-World Guide

What are the alternatives to using Pine script?

TD Ameritrade's thinkorswim – this platform has a lot of similarities to Pine Script.

It utilizes a proprietary language called thinkScript and stores price data in arrays in a similar way to Pine script.

Data is built-in and the platform is geared toward creating custom indicators and strategies.

Ninjatrader – This platform also uses a proprietary language which is called Ninjascript. The language is not completely proprietary as it is based on C#.

Ninjatrader has a bit more flexibility as it allows you to connect to custom data feeds.

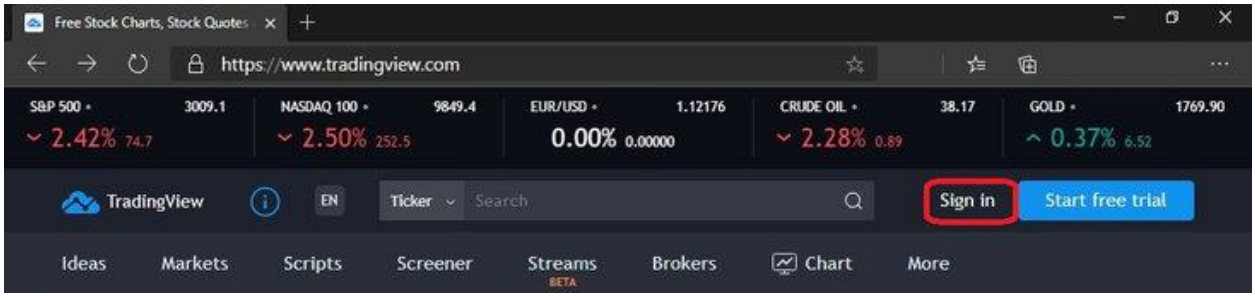
If you're already familiar with C#, C, or C++, this might be a viable alternative.

How do I get started with Pine script?

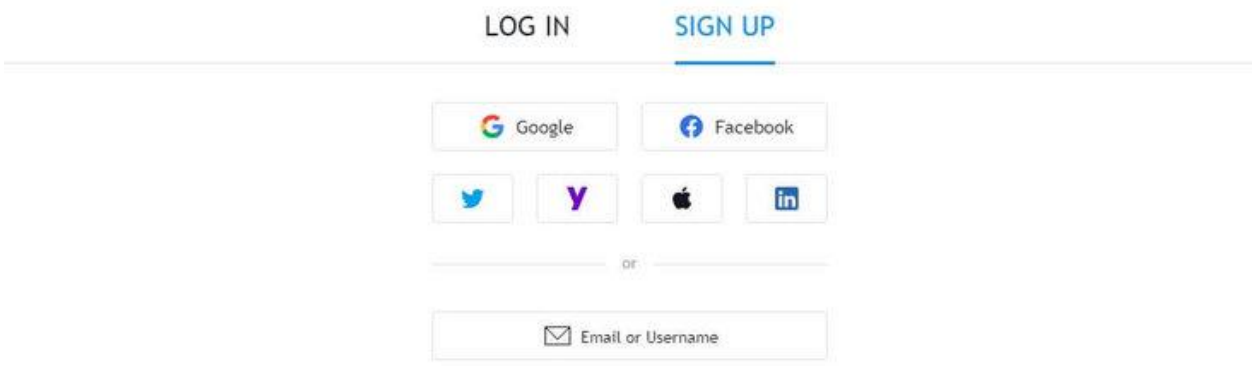
Getting started with Pine script is really simple, there is nothing to download or install.

If you already have an account with TradingView, simply head over to their page.

If you don't have an account, navigate to www.tradingview.com. From there you will see a sign-in box in the upper right-hand corner.



There are several one-click options to sign up, or use the traditional email/password method.



Having an account allows you to save your scripts to the TradingView cloud, and provides the ability to add custom indicators to your charts.

Once signed up, launch the charting platform either by clicking on chart in the menu or by navigating to www.tradingview.com/chart

A first look at Pine editor

Pine editor is where we will be creating our code. To launch it, click on Pine Editor on the very bottom of your screen.

A screen should pop up that looks like the image below.

In Pine script, you will either be creating an indicator or a strategy. We will discuss the differences extensively in this article.

If you're following along, the screen you're looking at now is the default starting script to create an indicator.

Let's run through it line by line.

```
// This source code is subject to the terms of the Mozilla Public License 2.0 at https://mozilla.org/MPL/2.0/
```

The first line is simply a comment. Comments in Pine script start with two forward slashes.

```
// © author
```

The second line is also a comment, it is auto-populated with your TradingView user name.

```
//@version=5
```

On the fourth line, you might assume we have yet another comment. However, this line is a bit different.

This is known as a compiler directive. It lets the compiler know which version of Pine script we want to use.

You can forego the first two comment lines if you want, but the compiler directive is required in all scripts.

```
indicator("My Script")
```

Line 5 is a declaration. This is where you specify if you are creating an indicator.

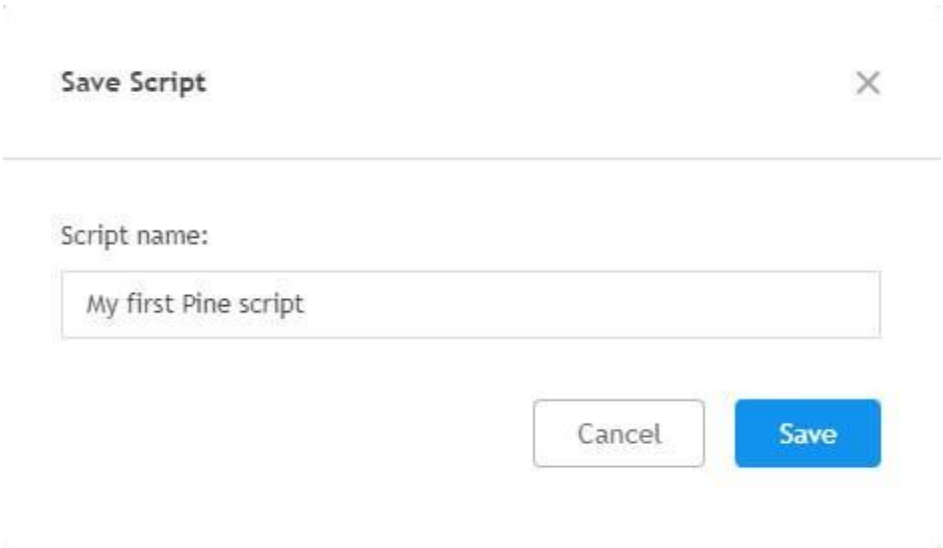
The alternative is to create a strategy, but we will start with the indicator.

The second part of Line 5, in quotation marks, is the name that we will assign for this particular indicator. The default is My Script.

`plot(close)`
Line 6 contains the plot command. As you may have guessed, this tells TradingView to plot a specific variable.

In this case, the variable `close` will get plotted. This is a built-in variable that contains the closing price of the latest bar.

Let's hit Add to Chart on the upper right of the Pine editor.



You will be prompted to save the script.

Once saved, your chart should have a new window that contains a plot of the closing prices from your main chart.

This window is called the data window. In the image above, this is the line chart that is drawn in blue.

Note that the data window shows "My Script" in the upper left-hand corner. This pulls whatever is entered into Line 5 of our code where we declared a name for the indicator.

And there you have it, our first indicator and we didn't even need to write any code!

How to retrieve the price of Apple?

In our first example, we plotted the closing price. Pine script will automatically do that for whichever chart you have open.

In this case, we had a daily chart of Bitcoin open.

But what if you want to get data for another asset? Let's go through an example where we grab the price of Apple even though we don't have its chart open.

Every script will start with a few lines where we set the compiler directive. We also indicate if it's an indicator or strategy that we are creating, and assign a name.

```
//@version=5
indicator("price of Apple")
```
In this case, we are creating an indicator. The name of this indicator is “price of Apple”.

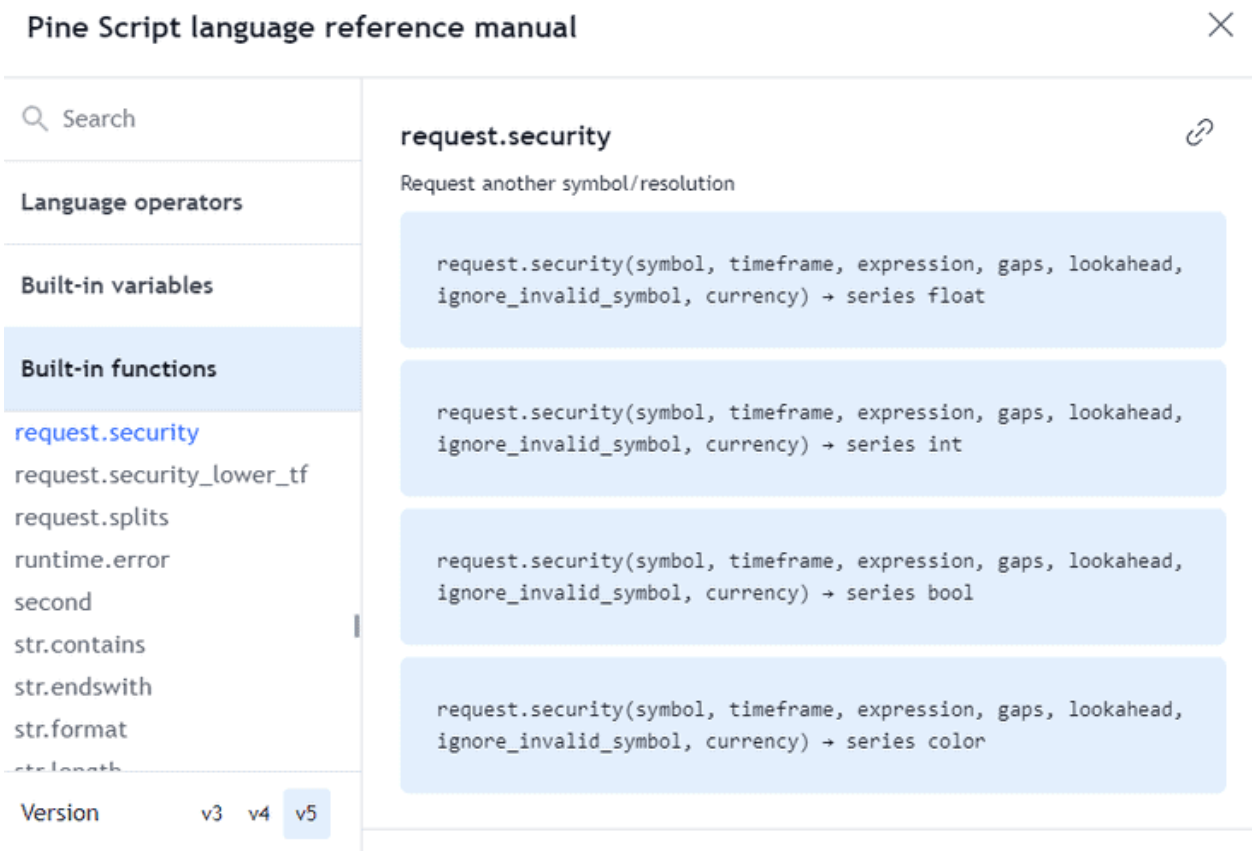
Next, we have to tell Pine Script that we are interested in an asset other than what is currently displayed on the chart.

To do this, we can use the request.security() function.

```
request.security()
```
A nice feature of Pine script is that help is always easily available if you’re working with the syntax you haven’t worked with before.

For example, we can hover over our function and it will show a brief description.

For more detailed information, you can launch a help window. To do this, hit CTRL while clicking on the function on a PC. Or, on a Mac, press CMD while clicking on the function.



The help function clarifies the syntax and even has helpful examples.

```
request.security("AAPL", "D", close)
```

We've used syntax similar to the example in the above code snippet. The first value in the security function is the ticker symbol which is AAPL.

Then we set the time frame to daily. And lastly, we told Pine script we are interested in the closing price.

We can save the return of the function to a variable.

```
apple_price = request.security("AAPL", "D", close)
```

Now the apple_price variable will contain the latest daily close of Apple's stock.

Studies created in Pine script need to have at least one output, otherwise, the script will generate a compiler error.

Let's plot our variable so that it satisfies the Pine script rule about having an output.

```
plot(apple_price)
```

After saving and adding to the chart, this is what our screen looks like.

We now have Apple's daily closing price plotted in the data window while the main window is showing a candlestick chart of Bitcoin.

Full Code:

```
//@version=5
indicator("price of Apple")

apple_price = request.security("AAPL", "D", close)

plot(apple_price)
```

How to retrieve the SMA(20) of Apple?

Now that we can access Apple's stock price, let's go through an example of retrieving a simple moving average.

The same process can be used to apply any indicator.

We will start with our basic declarations and use the security function we created in our last example.

```
//@version=5
```

```
indicator("Get 20 SMA of Apple")

// Get price of Apple
apple_price = request.security("AAPL", "D", close) // 1 Day
```
If you're not looking to get the 20 SMA specifically for AAPL, you can skip the security definition and just use the built-in `close` variable.

This will grab the closing price for whichever security you have showing in your main chart window.

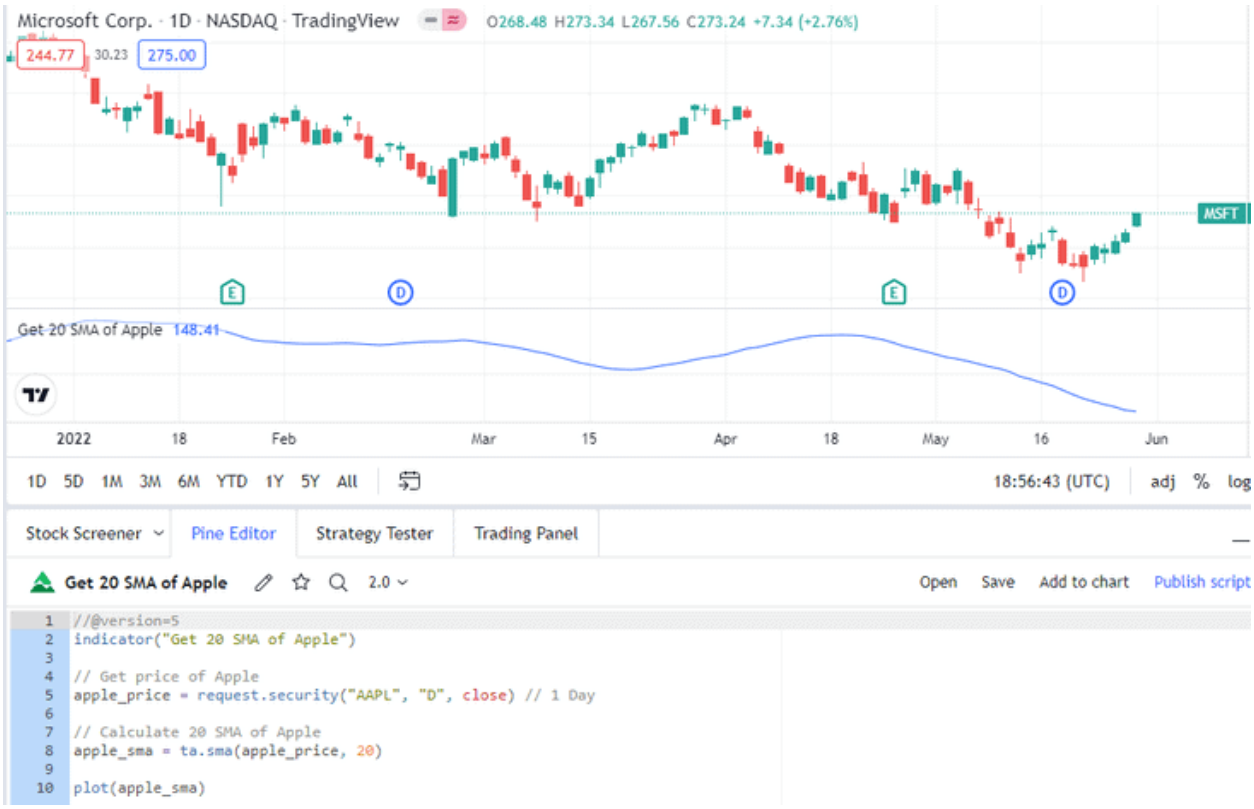
There is a helper function for the SMA indicator built-in to Pine script. To access it, we simply use the ta.sma() function.

```
ta.sma(apple_price, 20)
```
The first parameter we need to pass in is the price value. In this case, we are using the closing price for Apple that we have stored in our `apple_price` variable.

The second parameter is the length of the SMA. We are looking for a 20-period SMA.

Lastly, we will assign the SMA data to a separate variable and then plot it.

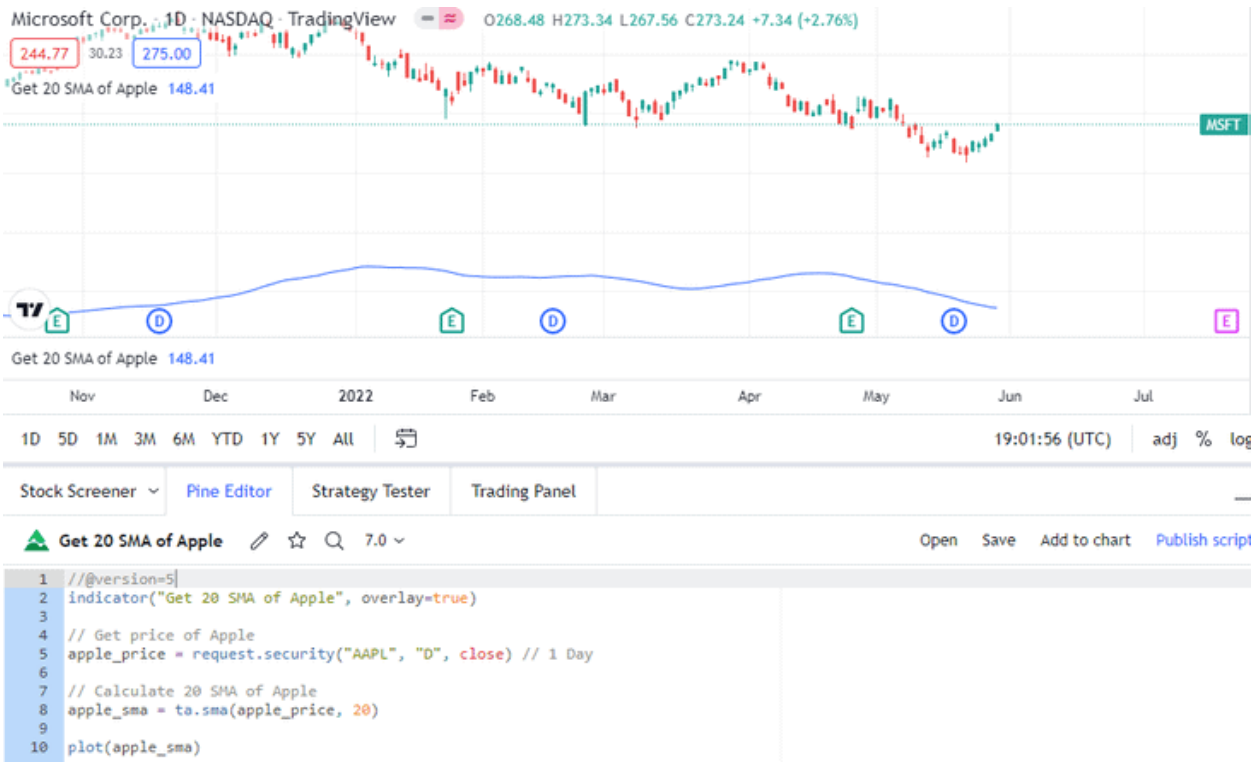
```
apple_sma = ta.sma(apple_price, 20)

plot(apple_sma)
```
The simple moving average for Apple is now plotted to our data window.



Moving averages are typically plotted on the main chart. We can achieve that with a slight modification in our code.

```
indicator("Get 20 SMA of Apple", overlay=true)
```

By adding in `overlay=True` into the indicator declaration, we can plot our data directly into the main charting window as opposed to the data window.



Full Code:

```
//@version=5
indicator("Get 20 SMA of Apple", overlay=true)

// Get price of Apple
apple_price = request.security("AAPL", "D", close) // 1 Day

// Calculate 20 SMA of Apple
apple_sma = ta.sma(apple_price, 20)

plot(apple_sma)
```

How to backtest a moving average cross strategy with Pine Script?

We've gone over indicators. Let's take a look at strategies in Pine Script.

In the next example, we will create a moving average cross-over strategy with a few additional parameters. We will then backtest the strategy within TradingView.

```
//@version=5
strategy("My Strategy", overlay=true)
```

To create a strategy, we swap out the indicator declaration with a strategy declaration.

```
// Create Indicator's
shortSMA = ta.sma(close, 10)
longSMA = ta.sma(close, 30)
```

The first thing we will want to do is create two moving averages and assign the data to variables.

```
rsi = ta.rsi(close, 14)
```

We will also create an RSI indicator that will be used to confirm our entries and exits.

This strategy will be run on the main chart so we don't need to use the security() function here.

Next, we want to specify our crossover conditions. Fortunately, TradingView has a built-in function for that already, so we don't need to code it manually.

```
// Specify crossover conditions
longCondition = ta.crossover(shortSMA, longSMA)
shortCondition = ta.crossunder(shortSMA, longSMA)
```

We have two conditions, the first one is when the short SMA, the 10-period, crosses above the longer 30-period SMA.

The second condition is the opposite as we've used the crossunder function as opposed to crossover.

Both these conditions are saved to variables. So when the crossover or crossunder occurs, these variables will get updated to True which is a Boolean value.

We can use an if statement to check if the condition is changed to True, and then execute a trade based if that is the case.

```
if (longCondition)
    strategy.entry("long", strategy.long, 100, when = rsi > 50)
```

The built-in `strategy.entry` function is used to enter trades. Here are the parameters that are passed into the function.

1. "long" – this is a trade ID. We won't be using it in this example. But, if you plan to close or cancel a trade, it can be done with this ID.
2. strategy.long – this is a built-in variable that tells Pine script that we want to get long.
3. 100 – the number of shares we want to trade
4. when = rsi > 50 – this is an additional parameter that tells pine script to only execute the trade if the RSI is higher than 50.

The syntax for our short entries will follow a very similar format.

```
if (shortCondition)
    strategy.entry("short", strategy.short, 100, when = rsi < 50)
```

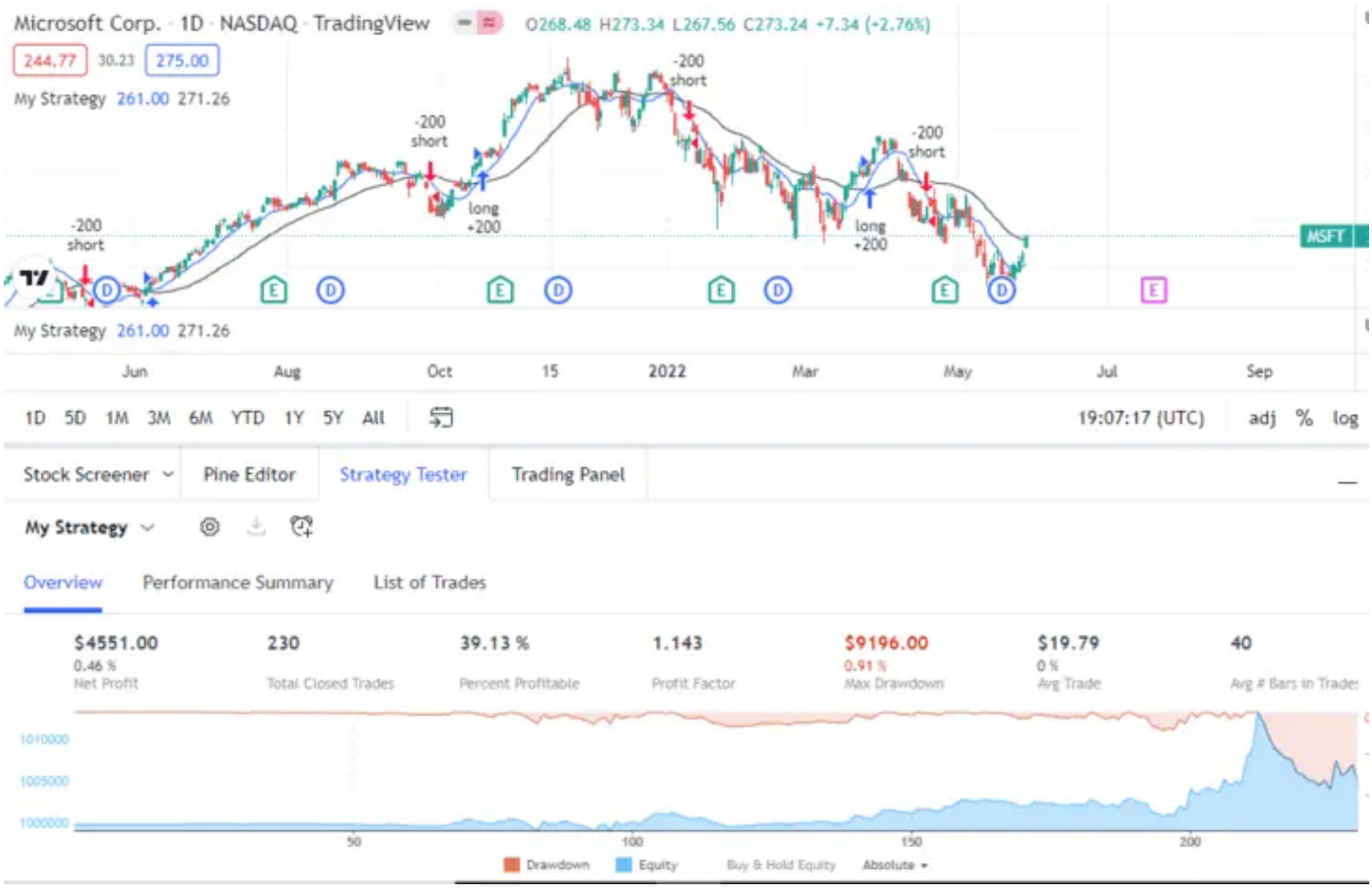
Since we are running a strategy, we don't have to plot anything or specify an output.

But we will do so anyway. It would be nice to see the SMA's on the chart so that we can confirm that trades took place when they should have.

```
// Plot Moving Average's to chart
plot(shortSMA)
plot(longSMA, color=color.black)
```

If we *save* and *add to chart*, the strategy will run and automatically open the Strategy Tester window which will display some important stats.

This is what our screen looks like.



By default, a new tab opens showing the overview stats for the strategy. You can click through the Performance Summary or List of Trades to see other statistics.

The strategy will run on the time frame that is displayed on your chart.

You can easily cycle through different time frames using the time frame options in the menu at the top of the screen. The strategy will auto-update based on the new time frame chosen.

Full Code:

```
//@version=5
strategy("My Strategy", overlay=true)

// Create Indicator's
shortSMA = ta.sma(close, 10)
longSMA = ta.sma(close, 30)
rsi = ta.rsi(close, 14)

// Specify crossover conditions
longCondition = ta.crossover(shortSMA, longSMA)
shortCondition = ta.crossunder(shortSMA, longSMA)
```

```
// Execute trade if condition is True
if (longCondition)
    strategy.entry("long", strategy.long, 100, when = rsi > 50)

if (shortCondition)
    strategy.entry("short", strategy.short, 100, when = rsi < 50)

// Plot Moving Average's to chart
plot(shortSMA)
plot(longSMA, color=color.black)
```

How to set take profits and stop losses?

In our last example, the trade execution was determined by moving average crossovers and crossunders.

We will build on this script and set specific stop losses and take profits. We can use the Average True Range (ATR) to calculate the levels for these.

The ATR indicator calculates the average movement over the last number of specified bars. This is a good way to account for changes in volatility.

We have already declared several indicators, we will add the ATR indicator to the list.

```
// Create Indicator's
shortSMA = ta.sma(close, 10)
longSMA = ta.sma(close, 30)
rsi = ta.rsi(close, 14)
atr = ta.atr(14)
```

Under our trade conditions, we can make the necessary calculations for our stop loss and take profit.

```
if (longCondition)
    stopLoss = low - atr * 2
    takeProfit = high + atr * 2
    strategy.entry("long", strategy.long, 100, when = rsi > 50)
    strategy.exit("exit", "long", stop=stopLoss, limit=takeProfit)
```

In the code above, we calculated the stop loss by taking the low of the bar at the time of entry and subtracting the average true range multiplied by two.

So if the stock moves on average $5 per bar, we are setting our take profit $10 below the low.

A similar calculation is done for the take profit.

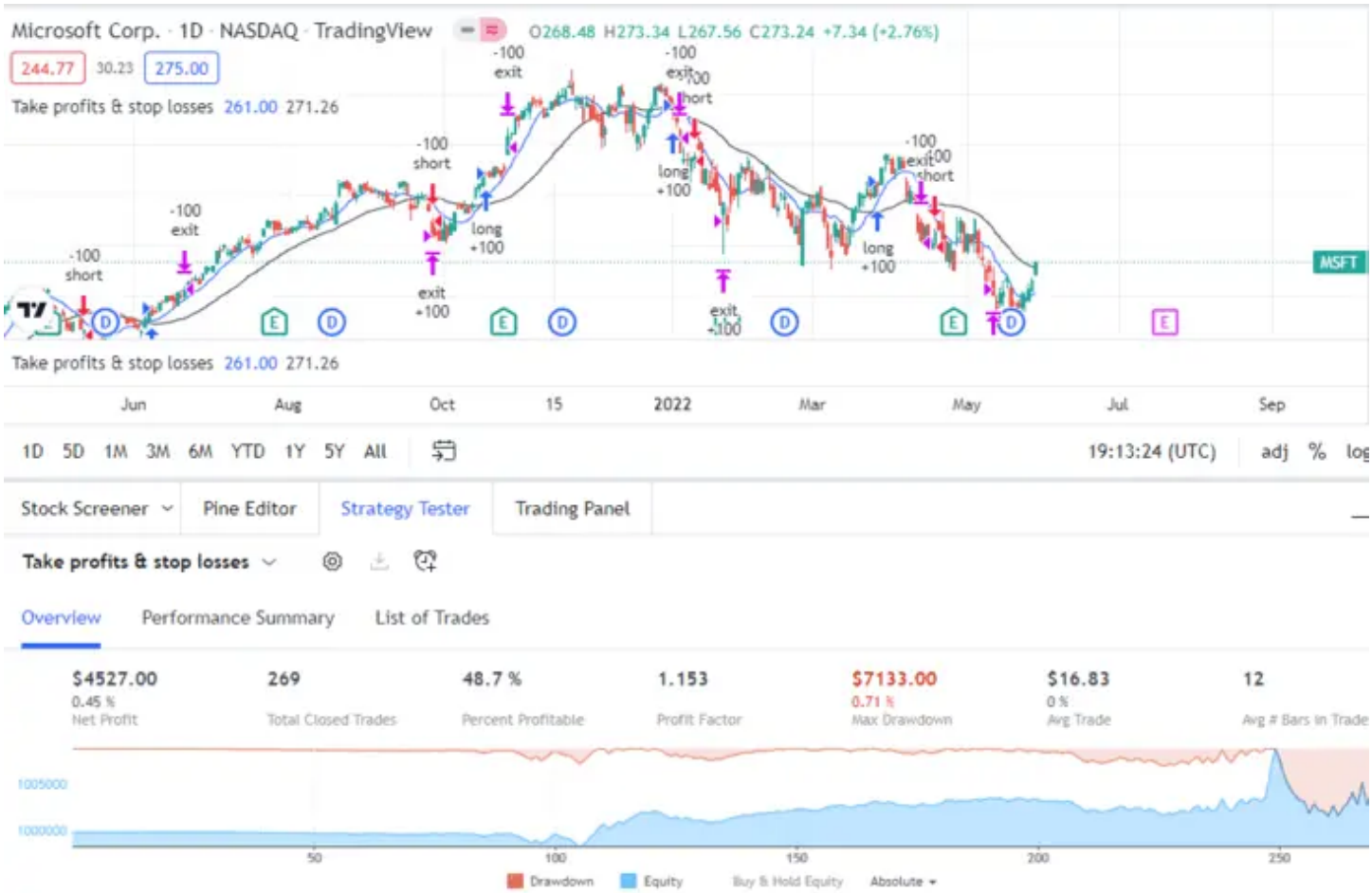
Lastly, we specify the exit condition using the strategy.exit() function. Here are the parameters that were passed through.

1. “exit” – this is the trade ID for exiting out of the trade.
2. ‘long” – this is the ID that we previously set when we entered the trade. This will let Pine script know which position we are trying to exit.
3. stop=stopLoss – we are specifying that the level contained in the stopLoss variable should be used as a stop order to exit the trade.
4. limit=takeProfit = we are specifying that the level contained in the takeProfit variable should be used as a limit order to exit the trade.

The syntax for our short condition is similar although some of the calculations are slightly different.

```
if (shortCondition)
    stopLoss = high + atr * 2
    takeProfit = low - atr * 2
    strategy.entry("short", strategy.short, 100, when = rsi < 50)
    strategy.exit("exit", "short", stop=stopLoss, limit=takeProfit)
```

The rest of the script remains unchanged from the prior example. Let’s run it and see how our strategy did.



Our exits are working and being plotted on our main chart along with the long and short entries.

Full Code:

```
//@version=5
strategy("Take profits & stop losses", overlay=true)

// Create Indicator's
shortSMA = ta.sma(close, 10)
longSMA = ta.sma(close, 30)
rsi = ta.rsi(close, 14)
atr = ta.atr(14)

// Specify crossover conditions
longCondition = ta.crossover(shortSMA, longSMA)
shortCondition = ta.crossunder(shortSMA, longSMA)

// Execute trade if condition is True
if (longCondition)
    stopLoss = low - atr * 2
    takeProfit = high + atr * 2
    strategy.entry("long", strategy.long, 100, when = rsi > 50)
    strategy.exit("exit", "long", stop=stopLoss, limit=takeProfit)

if (shortCondition)
    stopLoss = high + atr * 2
    takeProfit = low - atr * 2
    strategy.entry("short", strategy.short, 100, when = rsi < 50)
    strategy.exit("exit", "short", stop=stopLoss, limit=takeProfit)

// Plot Moving Average's to chart
plot(shortSMA)
plot(longSMA, color=color.black)
```

How to fire a trade on Apple when Google moves 5%?

We've seen that the security function can be used to display data for stocks not shown on the screen.

We will use it to create a strategy that will execute a trade in Apple if Google moves more than 5%.

This is a mean reversion strategy, so if Google rallies by more than 5%, we will short Apple. If Google falls by more than 5% then we can buy Apple.

The first thing we will do is store Google's daily open and closing price into a variable.

```
//@version=5
strategy("Pair Trade: Apple & Google")

google_close = request.security("GOOG", "D", close)
google_open = request.security("GOOG", "D", open)
```

We can then perform a calculation to determine the percentage price change.

```
price_change = google_close / google_open
```
The `price_change` variable now holds the calculation. So for example, if Google opened at $100 and rallied 5% to close at $105, the `price_change` variable would be 105/100 which is 1.05.

But if Google opened at $100, and declined 5% to close at $95, the variable would read 95/100 which is 0.95.

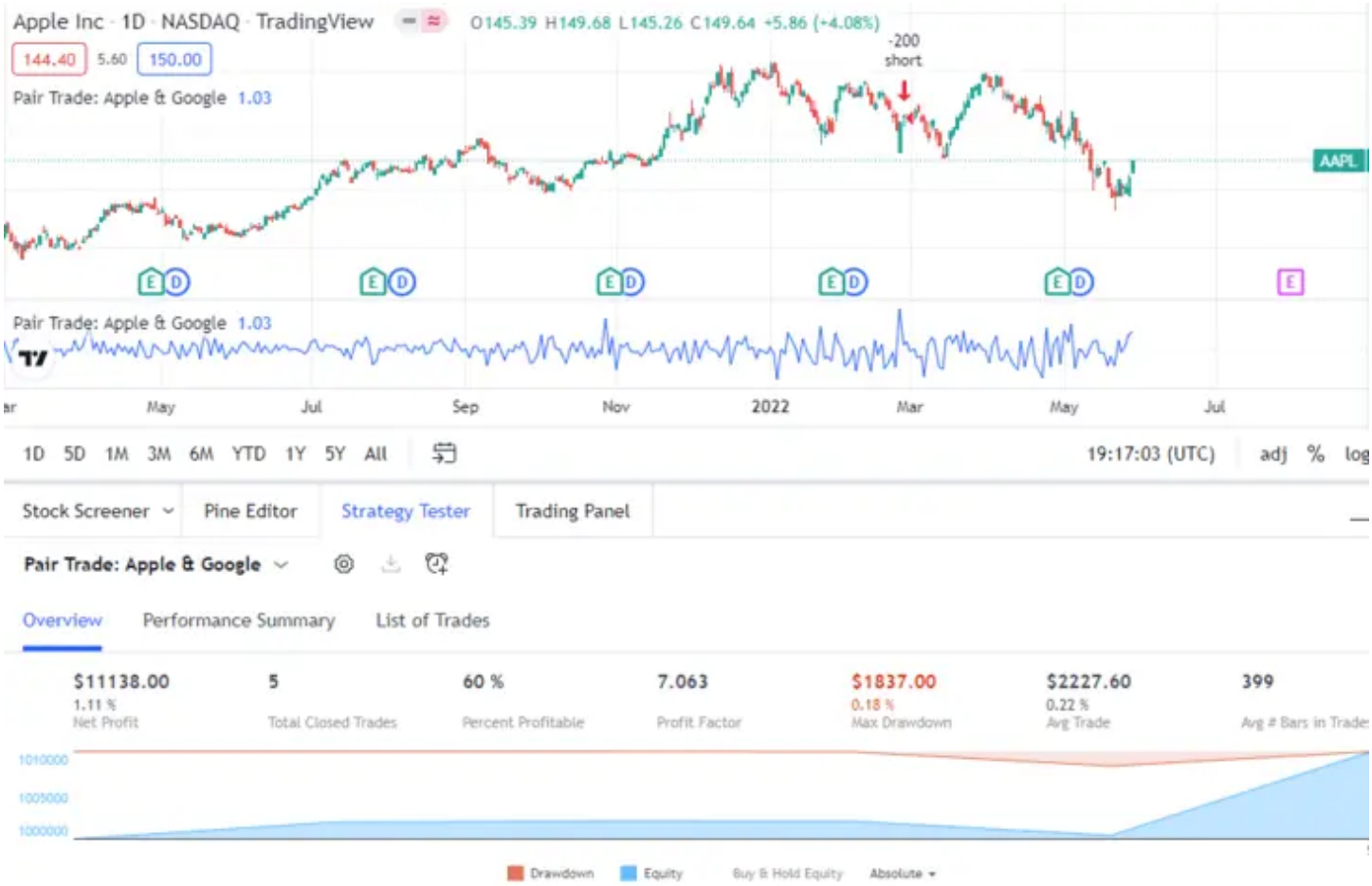
So we know that if Google declined 5% or more, the `price_change` variable would be 0.95 or less, and we want to get long. Here is the syntax to do that.

```
if price_change < 0.95
    strategy.entry("long", strategy.long, 100)
```
And the syntax to get short if Google rallies more than 5%.

```
if price_change > 1.05
    strategy.entry("short", strategy.short, 100)
```
You might notice that we have not mentioned Apple's stock price in the code. All we need to do is open an AAPL chart and it will automatically know to execute the trades in Apple.

Lastly, we will plot the `price_change` variable in the data window. It's not necessary, but nice to see and we can confirm that the trades are being executed as they should.

```
plot(price_change)
```
And here are the results of our strategy.



Only four trades as 5% movements are rare. We'd probably need to see a lot more trades than that to determine if it's a good strategy.

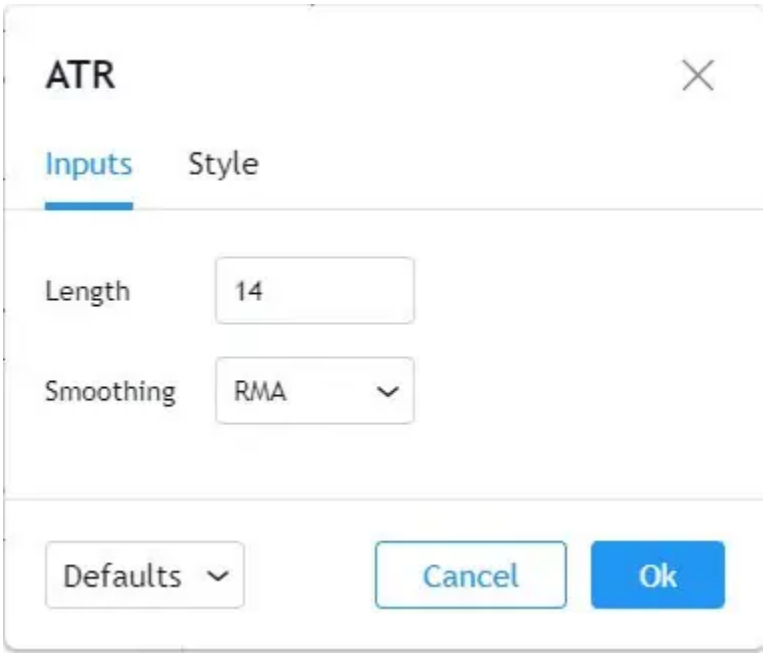
It did seem to have done a good job picking out that low in March!

Full Code:

```
//@version=5
strategy("Pair Trade: Apple & Google")

google_close = request.security("GOOG", "D", close)
google_open = request.security("GOOG", "D", open)

price_change = google_close / google_open

if price_change < 0.95
    strategy.entry("long", strategy.long, 100)

if price_change > 1.05
    strategy.entry("short", strategy.short, 100)

plot(price_change)
```

How to modify our scripts without coding?

A cool feature of Pine script is that we can create custom inputs to easily change the parameters of our strategies and indicators.



Take a look at the standard ATR indicator offered in Tradingview. Note how easy it is to modify the length and even the colors via the Style tab.

We can achieve the same for the studies and strategies created in Pine script by using the input() function.

Here is an example of the input function that will allow the user to customize the percent change from the last strategy example.

```
longTrigger = input(title="% Change for short entries", defval=5)

shortTrigger = input(title="% Change for long entries", defval=5)
```

Let's go through the parameters that are passed through the input() function.

1. title – this is where we specify the text that the user sees when trying to change the value of that particular parameter.
2. defval – this is the default value.

We can now get values from the user. But they will be inputting a value such as 5(%). We need to convert this to 1.05 for our if statements.

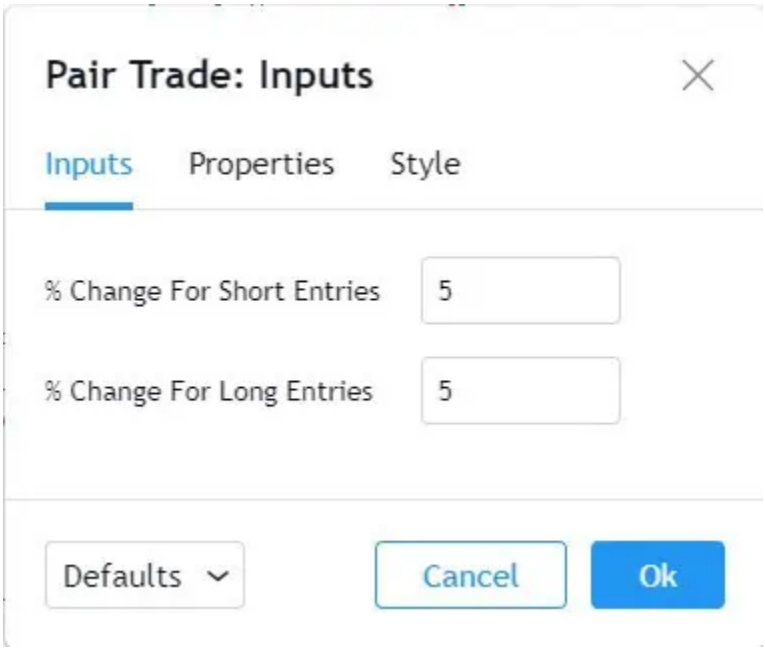
```
longTrigger := 1 - longTrigger/100
shortTrigger := 1 + shortTrigger/100
```

And we need to change our if statements to look at our newly created variables based on user input rather than the previously hard-coded values.

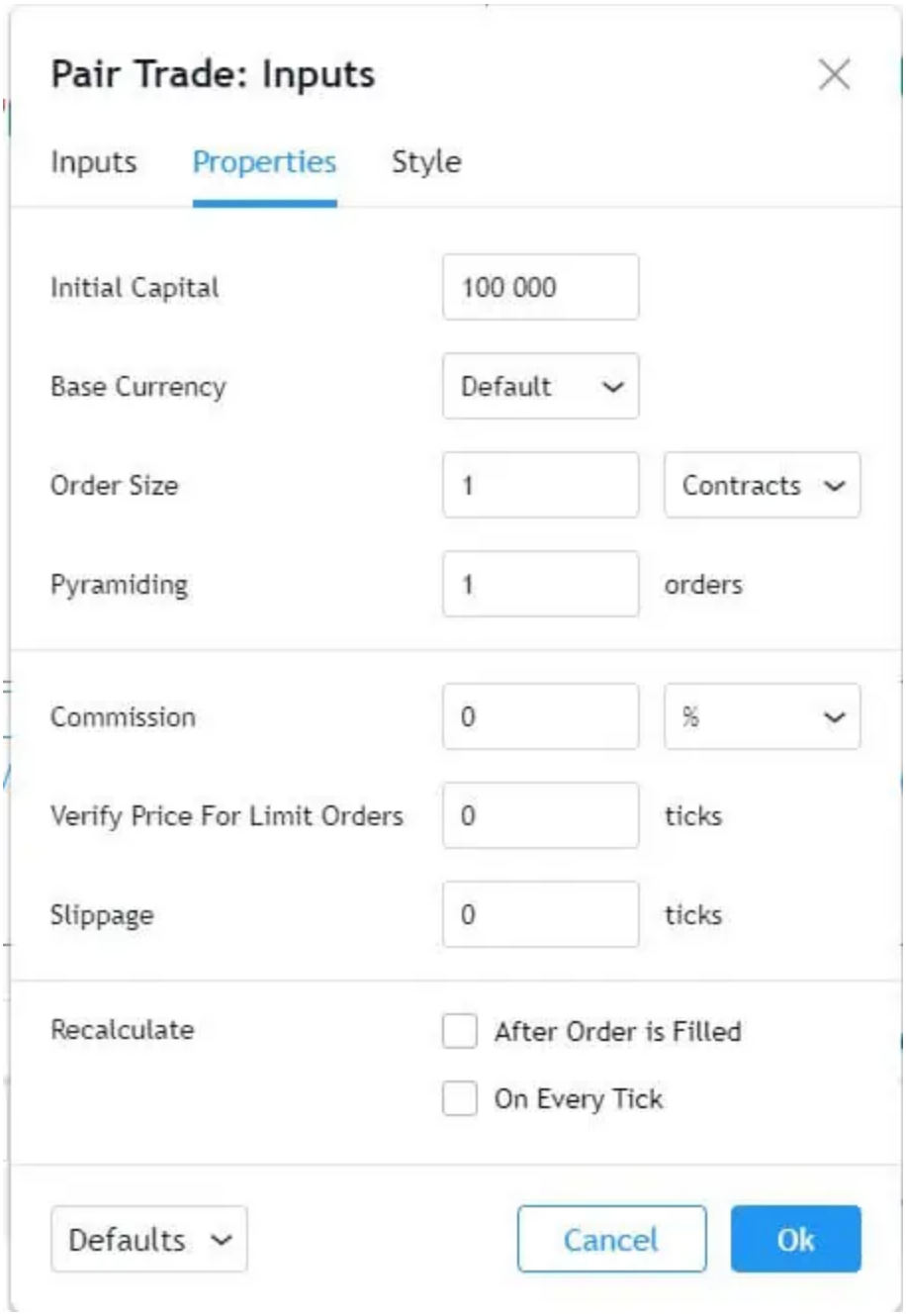
```
if price_change < longTrigger
    strategy.entry("long", strategy.long, 100)

if price_change > shortTrigger
    strategy.entry("short", strategy.short, 100)
```

To access the input options, click on the gear icon next to the name of your strategy in the data window.

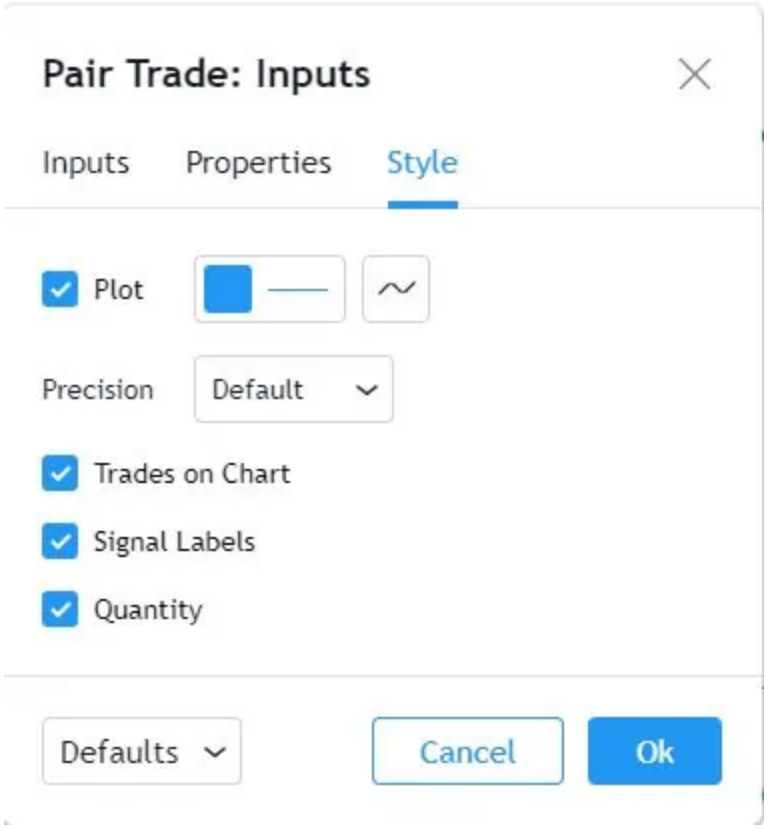
Custom values can now be set for the percentage change used in the strategy.



There is also a Properties window that will allow you to set custom options for other parts of the strategy.



And a Style window to customize plotting options.



Full Code:

```
//@version=5
strategy("Pair Trade: Inputs")

longTrigger = input(title="% Change for short entries", defval=5)
shortTrigger = input(title="% Change for long entries", defval=5)

longTrigger := 1 - longTrigger/100
shortTrigger := 1 + shortTrigger/100

google_close = request.security("GOOG", "D", close)
google_open = request.security("GOOG", "D", open)

price_change = google_close / google_open

if price_change < longTrigger
    strategy.entry("long", strategy.long, 100)

if price_change > shortTrigger
    strategy.entry("short", strategy.short, 100)

plot(price_change)
```

How to Plot with Pine script?

So far we've used the standard plot() function to plot certain things to the screen. Pine script has several other commands that we can use for our output and we will go through a few of them.

Plotting Forex market hours

Knowing when the markets open and close is something to be mindful of. This can be quite tough to figure out for Forex traders.

Forex trades 24 hours a day and 5 days a week. Different markets around the world open and close during the day which impacts currency volatility.

Let's program an indicator that will tell us with a quick glance at the chart when the markets are expected to be the busiest.

Most Forex traders are paying attention to the London and New York sessions. We will start by specifying the time for these sessions.

```
//@version=5
indicator("Forex Sessions", overlay=true)

London = time(timeframe.period, "0700-1500")
NY = time(timeframe.period, "1200-2000")
```

We've used the time() function here to create a period. In the parameters, we are using 0700 UTC for the start time of the London session and 1500 UTC for the end time.

The `London` variable will now contain the bar time if the bar falls in between that period. Otherwise, it will show a NaN (not a value).

We can use an if statement to see to check the output of the `London` variable.

```
val = if (na(London))
    1
else
    0
```

In the code above, we are using a built-in function called na(). What this does is check whether the variable has a NaN value or not.

if the `London`variable returns Nan, it means the bar is outside of London trading hours.

In this event, a variable called `val` will be assigned the integer 1. Otherwise, the `val`variable will be set at 0.

Lastly, we plot the newly created `val`variable.

```
plot(val)
```

Here is what our chart looks like after saving and adding this indicator to the chart.

That doesn't look too good. It is correctly showing when the London market is open, but plotting those values has made our candlesticks illegible.

We could plot it in the data window so that the candles are easier to see, but it still would not be easy to visualize the market open and close.

Also, the code doesn't look too good.

Let's start by using a one-line if statement to clean up our code a bit.

```
na(London) ? 1 : 0
```

This code performs the same function as the if statement before. We don't need to use the `val`variable in this case. Let's break down the syntax.

```
na(London)
```

This part is checking to see if the `London`variable contains a NaN value.

```
? 1 : 0
```

The question mark here is a short form for an if/else statement.

What follows the question mark is the important part. There are two numbers here separated by a colon. The number before the colon, 1 in this case, is what should be returned in the event the if statement is true.

The number after the colon, 0 in this case, gets returned when the if statement returns false.

We can then take the entire syntax and wrap it in a plot function, saving the effort of storing it to a variable first.

```
plot(na(London) ? 1 : 0)
```
So now we've cleaned up the if statement into a one-line piece of code.

To make the chart easier to read, we can plot a different background color if the London market is open.

To do this, we swap the plot() function with the bgcolor() function. This allows us to change the background color.

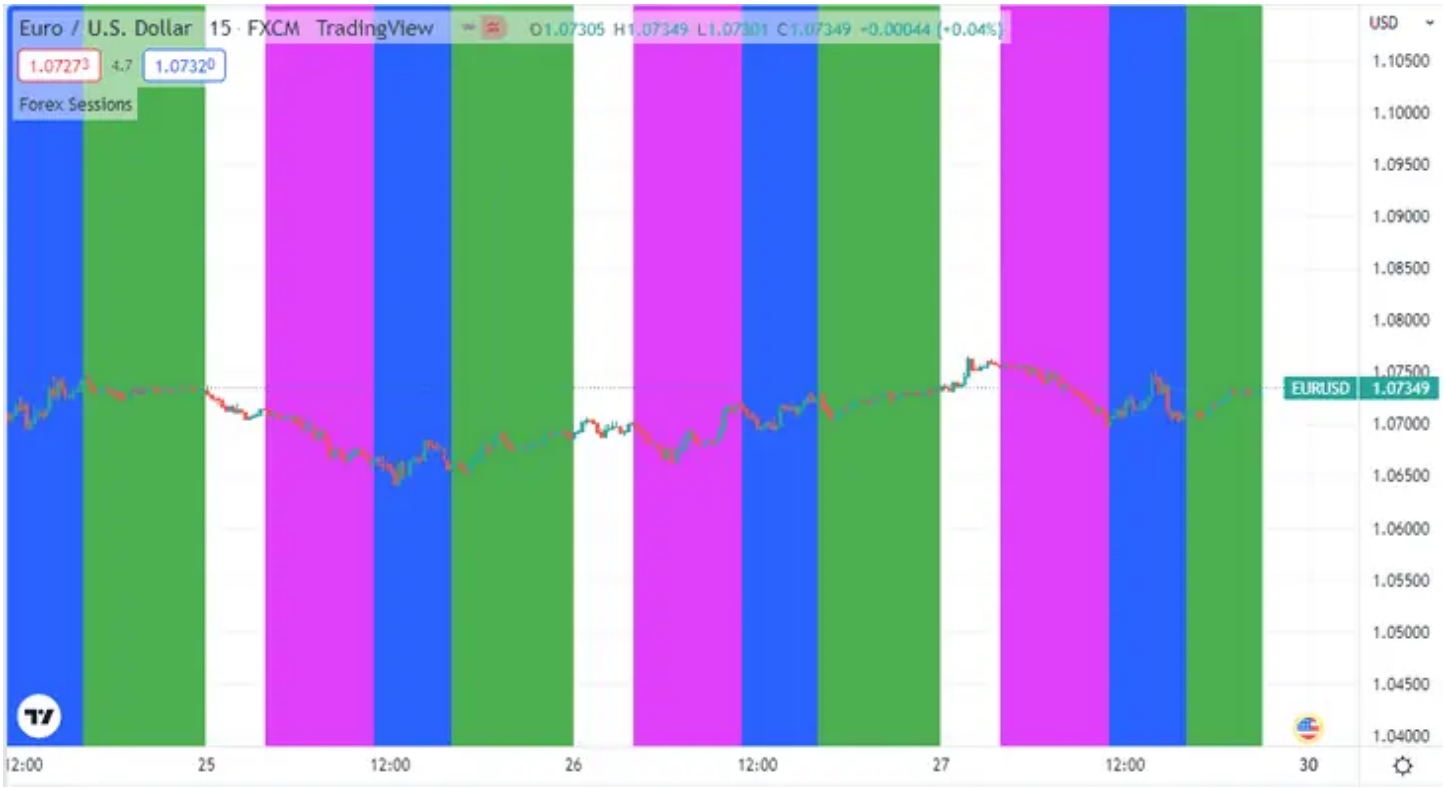
```
bgcolor(na(London) ? 1 : 0)
```
Also, we will specify a color for when the market is open.

```
bgcolor(na(London) ? na : color.blue)
```
Our chart is starting to look a lot better!

The last thing we will do is add code to see if the New York market is open, and set the background to green if it is.

```
bgcolor(na(NY) ? na : color.green)
```

You'll notice that there are three colors on the chart below.



That's because there is an overlap between the London and New York sessions, this is usually the most volatile time of the day.

Now we can easily see the sessions and quickly pick out things like the high set in European trading or the low that was printed during the overlap.

Full Code:

```
//@version=5
indicator("Forex Sessions", overlay=true)

Tokyo = time(timeframe.period, "0000-0800")
London = time(timeframe.period, "0700-1500")
NY = time(timeframe.period, "1200-2000")

bgcolor(na(Tokyo) ? na : color.fuchsia)
bgcolor(na(London) ? na : color.blue)
bgcolor(na(NY) ? na : color.green)
```

Plotting Annotations

There are several options to print annotations. As an example, you can use the hline() function to draw a horizontal level across the chart.

There is a plotchar() function that allows you to plot ASCII characters on your chart. This is often

used to plot a note either on top or on the bottom of the price bar.

Another common plotting function is plotshape() which allows you to plot various shapes. This one is quite popular as a lot of people use it to plot arrows on the top or bottom of bars to show buy or sell signals.

For a complete list of the various annotations available, check out the Annotations overview in the Pine script user manual.

How can I create a custom indicator in Pine script?

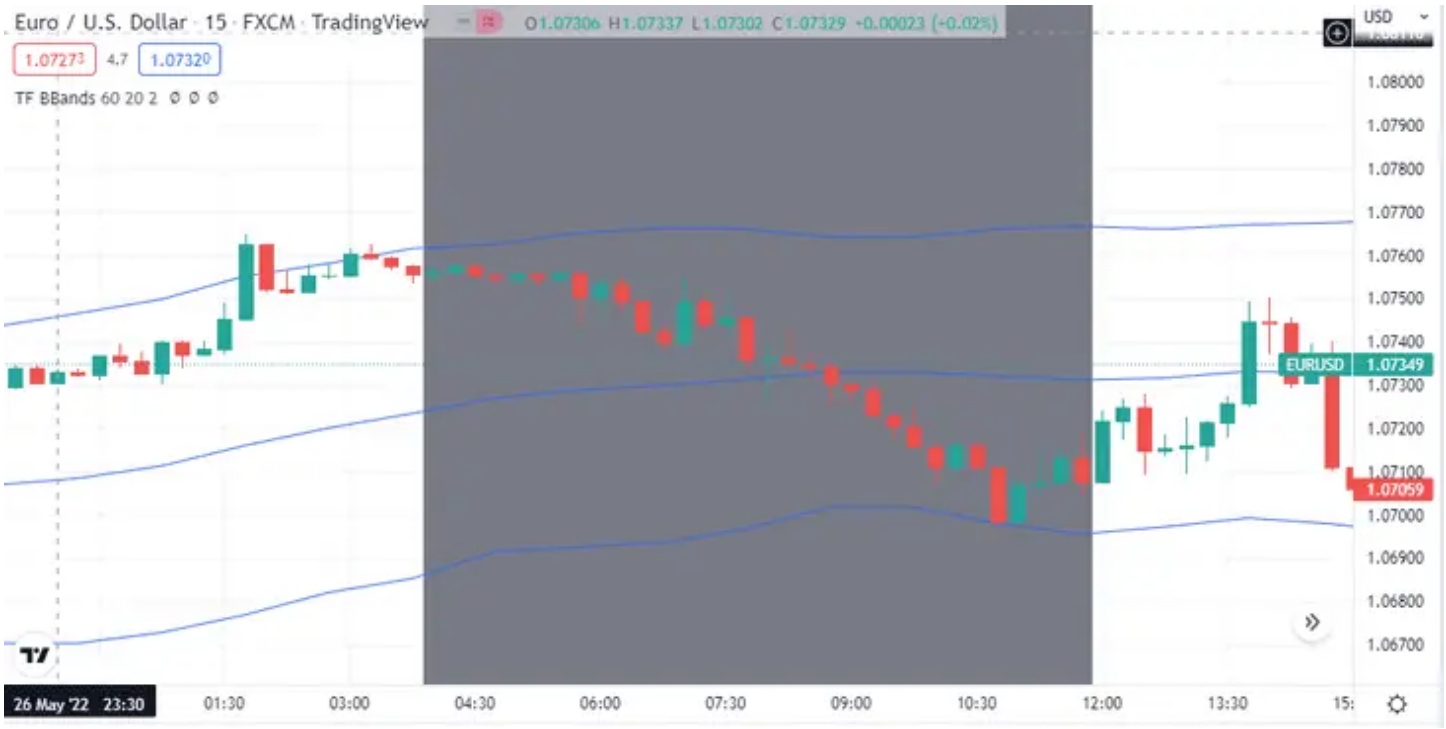
We are going to create a multi-timeframe indicator.

This is based on a scalping strategy that I used when I first started trading. It is a mean reversion strategy that works well during the early Asian session in the Forex markets when things are generally quiet.

The strategy uses Bollinger Bands on a 5-minute chart and RSI on a 1-minute chart.

The idea is to look for rsi divergence on a 1-minute chart when the price reaches the upper or lower Bollinger band on a 5-minute chart.

A potential target is the midline of the 5-minute Bollinger band or the lower line of a 1-minute Bollinger band.



The above image is an example of the strategy. The Forex sessions indicator that we used in a

previous example was used here to show when the Asian session is open.

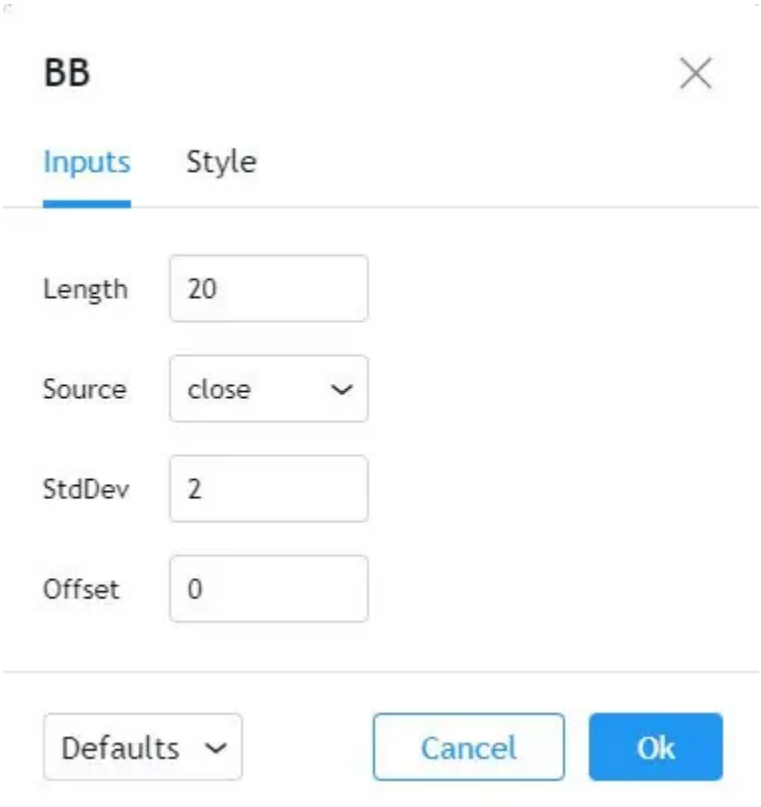
This strategy works best in the first half of the session, after that the risk of a breakout or directional move tends to increase.

When I traded this strategy, I had to keep two charts open, a 1-minute and a 5-minute chart.

But the example above shows the 5-minute Bollinger bands drawn directly on a 1-minute chart. We will create this indicator in Pine script.

```
//@version=5
indicator(title="Higher TF BBands", shorttitle="TF BBands",
overlay=true)
```

We start by declaring a name for the script and indicating it is an indicator. A shorter title can be added as well, this is the name that will be shown on the charts.



Next, we set some user inputs. We can duplicate most of the inputs from the regular Bollinger band indicator (as shown in the image above) for our custom indicator.

```
// Get user input
tf = input(title="BBands Timeframe", defval="5")
len = input(title="Length", defval=20)
stddev = input(title='StdDev', defval=2)
```

We can create the Bollinger band indicator from a built-in helper function.

```
[middle, upper, lower] = ta.bb(close, len, stddev)
```
There are three values returned from this function. The lower, mid, and upper band. These are saved individually to variables.

The values should be calculated on a different time frame. We can use the security() function to point to the time frame chosen by the user.

This is stored in the tf variable created by the earlier user input.

```
hbbandsMid = request.security(syminfo.tickerid, tf, middle,
barmerge.gaps_on, barmerge.lookahead_off)
hbbandsUpper = request.security(syminfo.tickerid, tf, upper,
barmerge.gaps_on, barmerge.lookahead_off)
hbbandsLower = request.security(syminfo.tickerid, tf, lower,
barmerge.gaps_on, barmerge.lookahead_off)
```
The ticker symbol remains the same, so we've used `syminfo.tickerid` which will return whichever ticker is being displayed on the main chart.

And that does it, all that's left is to plot the new indicator.

```
plot(hbbandsMid)
plot(hbbandsUpper)
plot(hbbandsLower)
```
We can now see Bollinger bands from a 5-minute chart displayed on a 1-minute chart.

The inputs allow for easy customization of Bollinger band parameters and allow this indicator to work with any time frame combination.

Full Code:

```
//@version=5
indicator(title="Higher TF BBands", shorttitle="TF BBands",
overlay=true)

// Get user input
tf = input(title="BBands Timeframe", defval="60")
len = input(title="Length", defval=20)
stddev = input(title='StdDev', defval=2)

[middle, upper, lower] = ta.bb(close, len, stddev)

hbbandsMid = request.security(syminfo.tickerid, tf, middle,
barmerge.gaps_on, barmerge.lookahead_off)
hbbandsUpper = request.security(syminfo.tickerid, tf, upper,
```

```
barmerge.gaps_on, barmerge.lookahead_off)
hbbandsLower = request.security(syminfo.tickerid, tf, lower,
barmerge.gaps_on, barmerge.lookahead_off)

plot(hbbandsMid)
plot(hbbandsUpper)
plot(hbbandsLower)
```

Final Thoughts

If you've been following along with the examples, you will have a good idea of what Pine script is capable of doing.

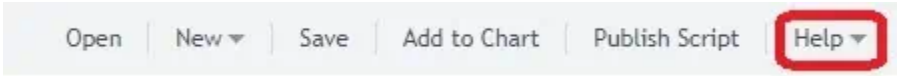
Overall, there is a lot you can do with Pine script, even though it has certain limitations. It's possible to code up a strategy really quickly once you get the hang of things.

From there, it's always an option to take that logic and program it into another language if you want to build on it and leverage third-party libraries.

The plotting functions are great, and the ability to make custom indicators is really useful for both manual traders and automated systems.

The comprehensive statistics offered for strategies is also a big plus point for Pine script.

TradingView has several resources if you want to take your Pine script coding skills a step further.



Some help functions have already been discussed in this article. In addition to that, there is also a help option from within Pine editor.

Here are a few other resources –

1. Quickstart Guide – This gives an example of an indicator and breaks the script down line by line.
2. Pine Script v5 User Manual – A detailed manual for Pine script.
3. TradingView Blog – Announcements of new features which often contain sample code
4. TradingView Scripts Library – A library of open source Pine script studies and strategies.

The last option on the list is a great resource as often another trader might have already coded the indicator or strategy you are after.

It is also a good resource to draw ideas from to build your own indicators or strategies.

If you'd like to try out some of the examples, a one-click download of all the code is available on GitHub. Simply click the green button and choose download zip. The code will be in text files which can be copied over to Tradingview's Pine editor.