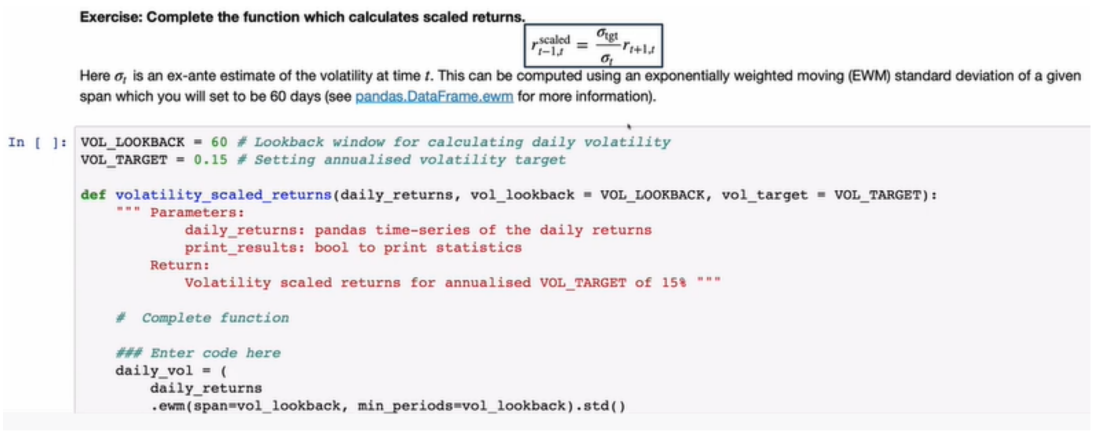
MODULE 4 UNIT 3
IDE Activity Video 1 Part 1 Transcript

Module 4 Unit 3 IDE Activity Video 1 Part 1 Transcript

STEFAN ZOHREN: So, what we do in the following we want to see how volatility targeting actually helps to improve scales performance.



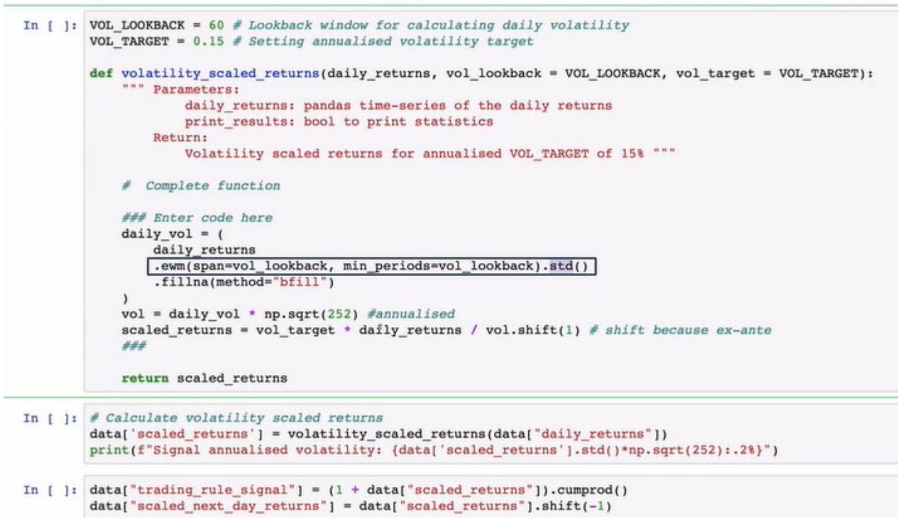
Exercise: Complete the function which calculates scaled returns.

$$\boxed{r^{\text{scaled}}_{t-1,t} = \frac{\sigma_{\text{tgt}}}{\sigma_t} r_{t+1,t}}$$

Here σ_t is an ex-ante estimate of the volatility at time t. This can be computed using an exponentially weighted moving (EWM) standard deviation of a given span which you will set to be 60 days (see pandas.DataFrame.ewm for more information).

```
In [ ]: VOL_LOOKBACK = 60 # Lookback window for calculating daily volatility
        VOL_TARGET = 0.15 # Setting annualised volatility target

        def volatility_scaled_returns(daily_returns, vol_lookback = VOL_LOOKBACK, vol_target = VOL_TARGET):
            """ Parameters:
                    daily_returns: pandas time-series of the daily returns
                    print_results: bool to print statistics
                Return:
                    Volatility scaled returns for annualised VOL_TARGET of 15% """

            #  Complete function

            ### Enter code here
            daily_vol = (
                daily_returns
                .ewm(span=vol_lookback, min_periods=vol_lookback).std()
```

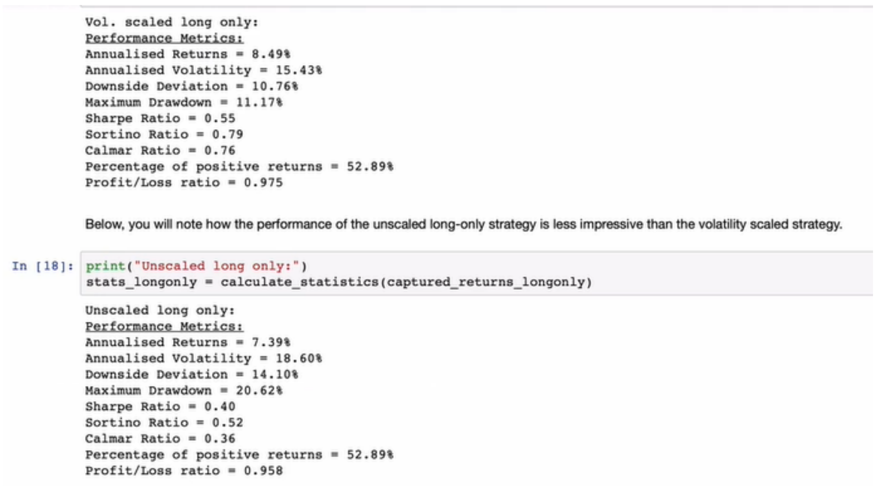
So here, volatility targeting just takes the returns, multiplies them times some target volatility, and then divide them by the actual volatility, or more precisely, a rolling estimate of this actual volatility, which is here done over a 60-day lookback window. So, we see the implementation is actually quite straightforward.



```
In [ ]: VOL_LOOKBACK = 60 # Lookback window for calculating daily volatility
        VOL_TARGET = 0.15 # Setting annualised volatility target

        def volatility_scaled_returns(daily_returns, vol_lookback = VOL_LOOKBACK, vol_target = VOL_TARGET):
            """ Parameters:
                    daily_returns: pandas time-series of the daily returns
                    print_results: bool to print statistics
                Return:
                    Volatility scaled returns for annualised VOL_TARGET of 15% """

            #  Complete function

            ### Enter code here
            daily_vol = (
                daily_returns
                .ewm(span=vol_lookback, min_periods=vol_lookback).std()
                .fillna(method="bfill")
            )
            vol = daily_vol * np.sqrt(252) #annualised
            scaled_returns = vol_target * daily_returns / vol.shift(1) # shift because ex-ante
            ###

            return scaled_returns

In [ ]: # Calculate volatility scaled returns
        data['scaled_returns'] = volatility_scaled_returns(data["daily_returns"])
        print(f"Signal annualised volatility: {data['scaled_returns'].std()*np.sqrt(252):.2%}")

In [ ]: data["trading_rule_signal"] = (1 + data["scaled_returns"]).cumprod()
        data["scaled_next_day_returns"] = data["scaled_returns"].shift(-1)
```

To compute the rolling average of daily volatility, we take the returns, we take the exponentially rated moving standard deviation, and then we fill in the NAs. Importantly, again, we have to standardise and multiply by the square root of 252 to get the correct volatilities there. Now the scaled returns are just the target volatility times the actual returns divided by our rolling estimate of the volatility.

So interestingly, the annualised volatility of the long-only strategy was before was around 18.6%, now actually went down to 15.49%. We can now also plot them and plot the cumulative returns and see the actual improvement in the strategy by eye.



```
Vol. scaled long only:
Performance Metrics:
Annualised Returns = 8.49%
Annualised Volatility = 15.43%
Downside Deviation = 10.76%
Maximum Drawdown = 11.17%
Sharpe Ratio = 0.55
Sortino Ratio = 0.79
Calmar Ratio = 0.76
Percentage of positive returns = 52.89%
Profit/Loss ratio = 0.975
```

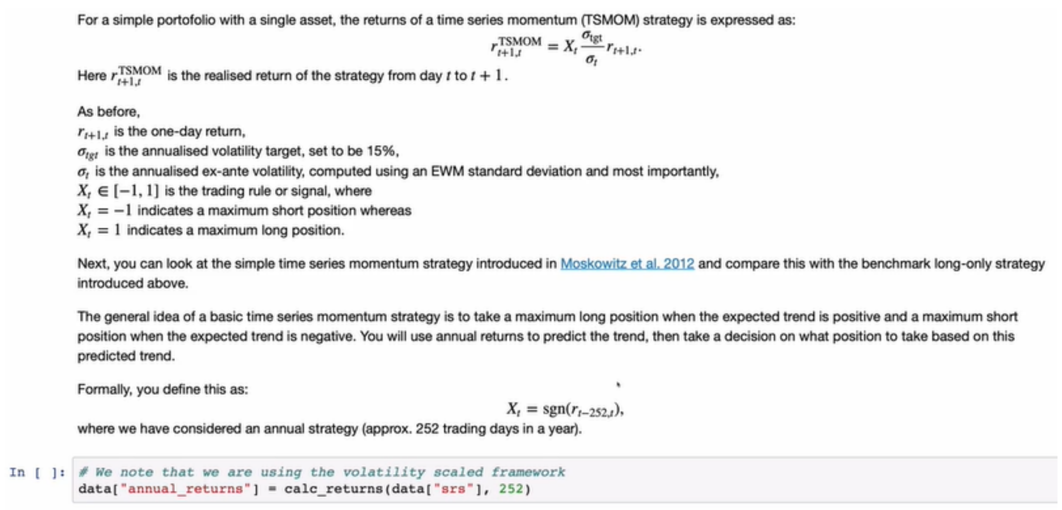
Below, you will note how the performance of the unscaled long-only strategy is less impressive than the volatility scaled strategy.

```
In [18]: print("Unscaled long only:")
         stats_longonly = calculate_statistics(captured_returns_longonly)
```

```
Unscaled long only:
Performance Metrics:
Annualised Returns = 7.39%
Annualised Volatility = 18.60%
Downside Deviation = 14.10%
Maximum Drawdown = 20.62%
Sharpe Ratio = 0.40
Sortino Ratio = 0.52
Calmar Ratio = 0.36
Percentage of positive returns = 52.89%
Profit/Loss ratio = 0.958
```

If you look at the metrics, we can also see that we see quite some significant improvements, for example: analyse return went up by nearly a percent, the volatility went down by nearly 3%, and importantly, the max drawdown which before was 20%, now became 11% by doing something as simple as just vol. targeting. So hopefully, this shows you the importance of doing such simple things and simple risk management, which can help to improve strategy performance.

In the next bit, we now move to time-series momentum. So, the simplest form of time-series momentum is a very easy model introduced by Moskowitz in 2012, which what it does is very simple. It just takes the last year's returns and says "if last year's returns were positive, I now go long over the next year. If last year's returns were negative, I now go short over the next year."



For a simple portofolio with a single asset, the returns of a time series momentum (TSMOM) strategy is expressed as:

$$r_{t+1,t}^{\mathrm{TSMOM}} = X_t \frac{\sigma_{\mathrm{tgt}}}{\sigma_t} r_{t+1,t}.$$

Here $r_{t+1,t}^{\mathrm{TSMOM}}$ is the realised return of the strategy from day t to $t+1$.

As before,
$r_{t+1,t}$ is the one-day return,
σ_{tgt} is the annualised volatility target, set to be 15%,
σ_t is the annualised ex-ante volatility, computed using an EWM standard deviation and most importantly,
$X_t \in [-1, 1]$ is the trading rule or signal, where
$X_t = -1$ indicates a maximum short position whereas
$X_t = 1$ indicates a maximum long position.

Next, you can look at the simple time series momentum strategy introduced in Moskowitz et al. 2012 and compare this with the benchmark long-only strategy introduced above.

The general idea of a basic time series momentum strategy is to take a maximum long position when the expected trend is positive and a maximum short position when the expected trend is negative. You will use annual returns to predict the trend, then take a decision on what position to take based on this predicted trend.

Formally, you define this as:

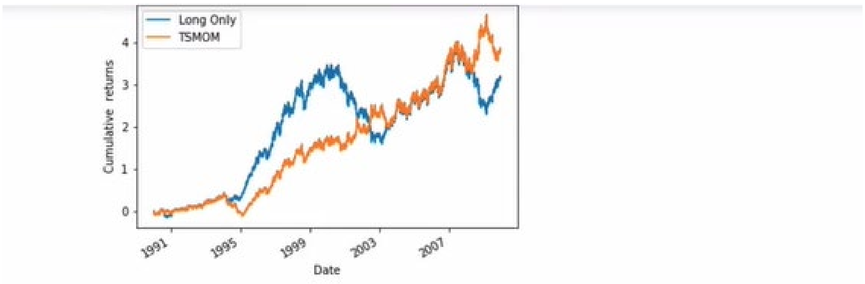
$$X_t = \mathrm{sgn}(r_{t-252,t}),$$

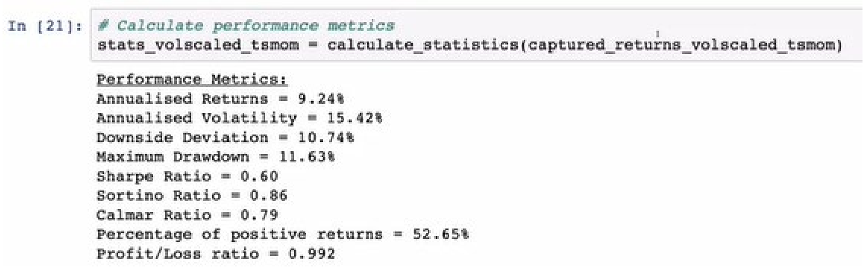
where we have considered an annual strategy (approx. 252 trading days in a year).

```
In [ ]: # We note that we are using the volatility scaled framework
        data["annual_returns"] = calc_returns(data["srs"], 252)
```

IN COLLABORATION WITH getsmarter™

That's a very simple form of momentum, but we see if it takes this momentum strategy and we additionally apply volatility targeting to it, because we have seen that volatility targeting is very helpful, then this will actually be a quite good strategy.

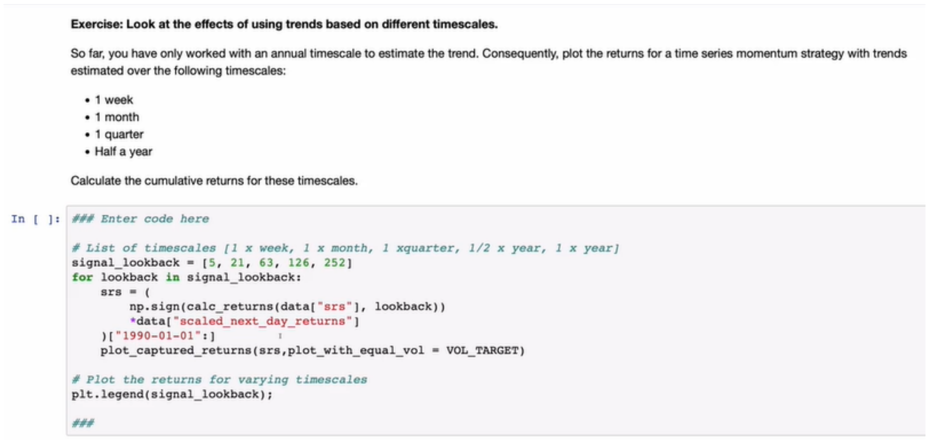




```
In [21]: # Calculate performance metrics
         stats_volscaled_tsmom = calculate_statistics(captured_returns_volscaled_tsmom)

         Performance Metrics:
         Annualised Returns = 9.24%
         Annualised Volatility = 15.42%
         Downside Deviation = 10.74%
         Maximum Drawdown = 11.63%
         Sharpe Ratio = 0.60
         Sortino Ratio = 0.86
         Calmar Ratio = 0.79
         Percentage of positive returns = 52.65%
         Profit/Loss ratio = 0.992
```

So here, we can actually see the performance of this strategy, and you can see the line is pretty nice and straight. So, it has—it looks by eye already a little bit better than the long-only strategy. And if you see if you look at the statistics, we see that our performance actually went further up. Now we have 9.24%, and we have annualised volatility of 15.42% because we were targeting a volatility of 15%, so we're doing quite good here. And drawdown also only around 11%. So, we see that this strategy actually performs quite well with a sharp ratio of 0.6, quite a bit up from the previous long-only strategy, even the long-only vol, targeting strategy, we are a little bit up.

So now in the exercise, what we want to do, is look at some different time periods. So, in the traditional strategy, we only looked at the last year.



Exercise: Look at the effects of using trends based on different timescales.

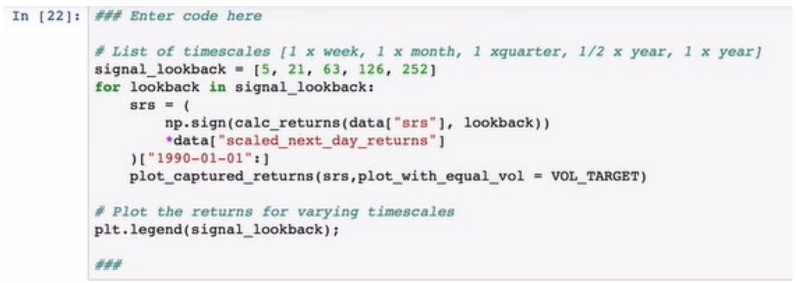
So far, you have only worked with an annual timescale to estimate the trend. Consequently, plot the returns for a time series momentum strategy with trends estimated over the following timescales:

- 1 week
- 1 month
- 1 quarter
- Half a year

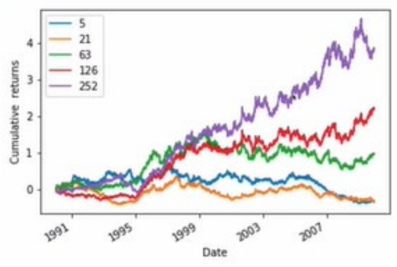
Calculate the cumulative returns for these timescales.

```
In [ ]: ### Enter code here

        # List of timescales [1 x week, 1 x month, 1 xquarter, 1/2 x year, 1 x year]
        signal_lookback = [5, 21, 63, 126, 252]
        for lookback in signal_lookback:
            srs = (
                np.sign(calc_returns(data["srs"], lookback))
                *data["scaled_next_day_returns"]
            )["1990-01-01":]
            plot_captured_returns(srs,plot_with_equal_vol = VOL_TARGET)

        # Plot the returns for varying timescales
        plt.legend(signal_lookback);

        ###
```

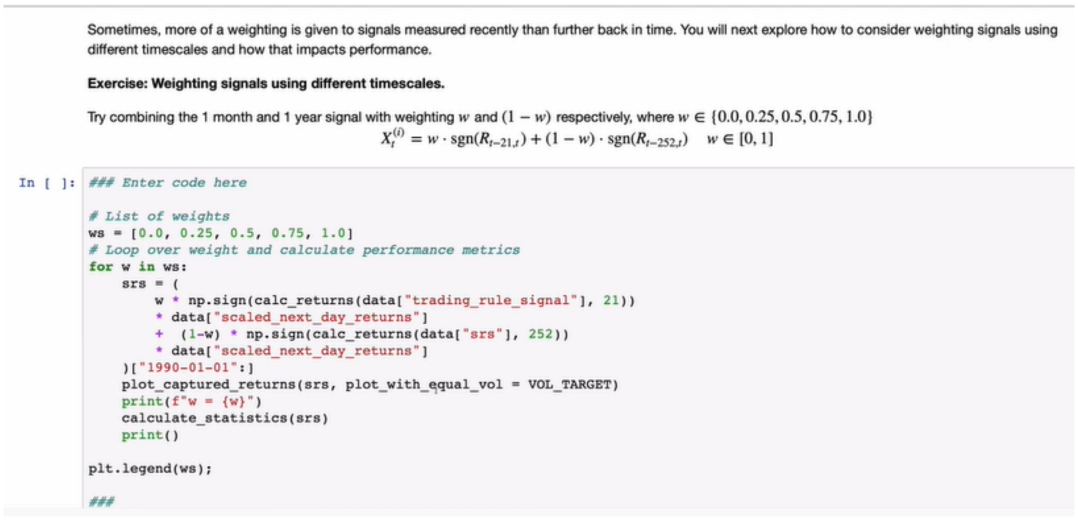
Now, we can also see what happens if we actually modify this horizon to say half a year, quarter, month or week. So, this is done in the following exercise, and we just define a lookback window here, and then we can just for each lookback, we can just compute the signal and then plot the performance of this strategy.



```
In [22]: ### Enter code here

         # List of timescales [1 x week, 1 x month, 1 xquarter, 1/2 x year, 1 x year]
         signal_lookback = [5, 21, 63, 126, 252]
         for lookback in signal_lookback:
             srs = (
                 np.sign(calc_returns(data["srs"], lookback))
                 *data["scaled_next_day_returns"]
             )["1990-01-01":]
             plot_captured_returns(srs,plot_with_equal_vol = VOL_TARGET)

         # Plot the returns for varying timescales
         plt.legend(signal_lookback);

         ###
```



And we can see here that actually the annual version of the strategy was the best performing one, and shorter horizon strategies actually perform worse with the kind of half year worse than one year, but still the best of the other ones. And the shorter we go, the worse it gets, basically.



Sometimes, more of a weighting is given to signals measured recently than further back in time. You will next explore how to consider weighting signals using different timescales and how that impacts performance.

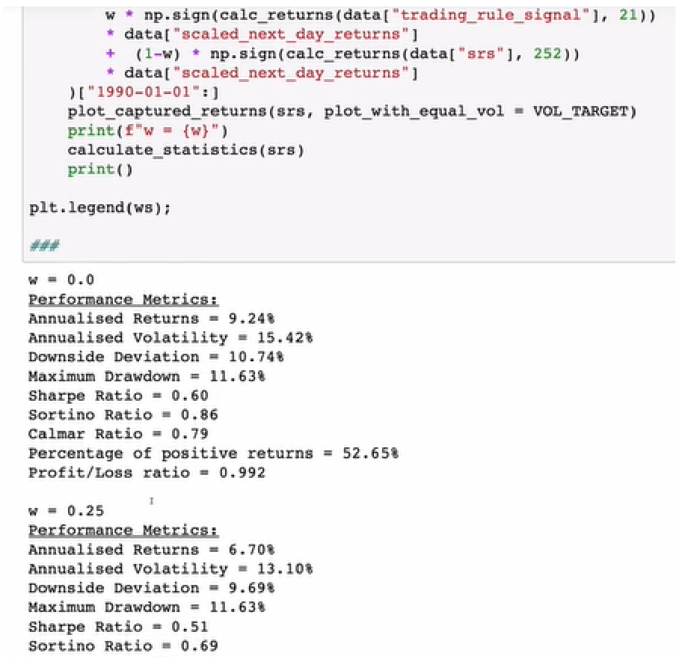
Exercise: Weighting signals using different timescales.

Try combining the 1 month and 1 year signal with weighting w and $(1-w)$ respectively, where $w \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$

$$X_t^{(i)} = w \cdot \mathrm{sgn}(R_{t-21,t}) + (1-w) \cdot \mathrm{sgn}(R_{t-252,t}) \quad w \in [0, 1]$$

```
In [ ]: ### Enter code here

        # List of weights
        ws = [0.0, 0.25, 0.5, 0.75, 1.0]
        # Loop over weight and calculate performance metrics
        for w in ws:
            srs = (
                w * np.sign(calc_returns(data["trading_rule_signal"], 21))
                * data["scaled_next_day_returns"]
                +  (1-w) * np.sign(calc_returns(data["srs"], 252))
                * data["scaled_next_day_returns"]
            )["1990-01-01":]
            plot_captured_returns(srs, plot_with_equal_vol = VOL_TARGET)
            print(f"w = {w}")
            calculate_statistics(srs)
            print()

        plt.legend(ws);

        ###
```

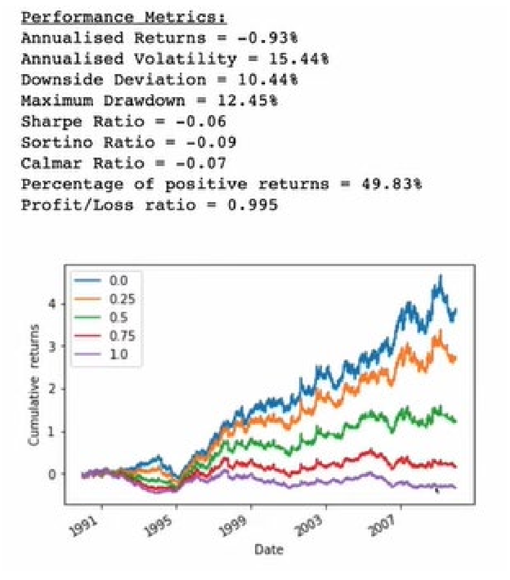
Now, in the next exercise, we wanted to see what happens if you actually combine some signals, so we have a kind of combination, a weighted combination of signals here, and to see whether that gives any improvement. In this case, we can look at weighting combinations between the one month and one year time-series momentum strategy, and we can weight them with different factors; either we have zero weighting of the one month or a quarter, half, three quarters or one, and we can look at the performance of that. So here, we fill in the details by just combining the weighted signals, and we just plot performance and the returns.



```
            w * np.sign(calc_returns(data["trading_rule_signal"], 21))
            * data["scaled_next_day_returns"]
            +  (1-w) * np.sign(calc_returns(data["srs"], 252))
            * data["scaled_next_day_returns"]
        )["1990-01-01":]
        plot_captured_returns(srs, plot_with_equal_vol = VOL_TARGET)
        print(f"w = {w}")
        calculate_statistics(srs)
        print()

plt.legend(ws);

###
```

```
w = 0.0
Performance Metrics:
Annualised Returns = 9.24%
Annualised Volatility = 15.42%
Downside Deviation = 10.74%
Maximum Drawdown = 11.63%
Sharpe Ratio = 0.60
Sortino Ratio = 0.86
Calmar Ratio = 0.79
Percentage of positive returns = 52.65%
Profit/Loss ratio = 0.992

w = 0.25
Performance Metrics:
Annualised Returns = 6.70%
Annualised Volatility = 13.10%
Downside Deviation = 9.69%
Maximum Drawdown = 11.63%
Sharpe Ratio = 0.51
Sortino Ratio = 0.69
```

And we can see it’s easiest to look at the plots that actually the best performing one was the annual one, and if we mix in the shorter version, it becomes—the performance becomes worse the more we mix in the one-month strategies there. So, the one-month strategy is not very helpful.



```
Performance Metrics:
Annualised Returns = -0.93%
Annualised Volatility = 15.44%
Downside Deviation = 10.44%
Maximum Drawdown = 12.45%
Sharpe Ratio = -0.06
Sortino Ratio = -0.09
Calmar Ratio = -0.07
Percentage of positive returns = 49.83%
Profit/Loss ratio = 0.995
```

So, this was basically all I wanted to say about the time-series. Let's just now look at a different version of momentum strategy, which is the moving average convergence divergence or MACD strategy.

SPEAKER: If you would like to review any of these sections, please click on the relevant button.