Machine Learning Trading

FinTech

Lesson 15.3

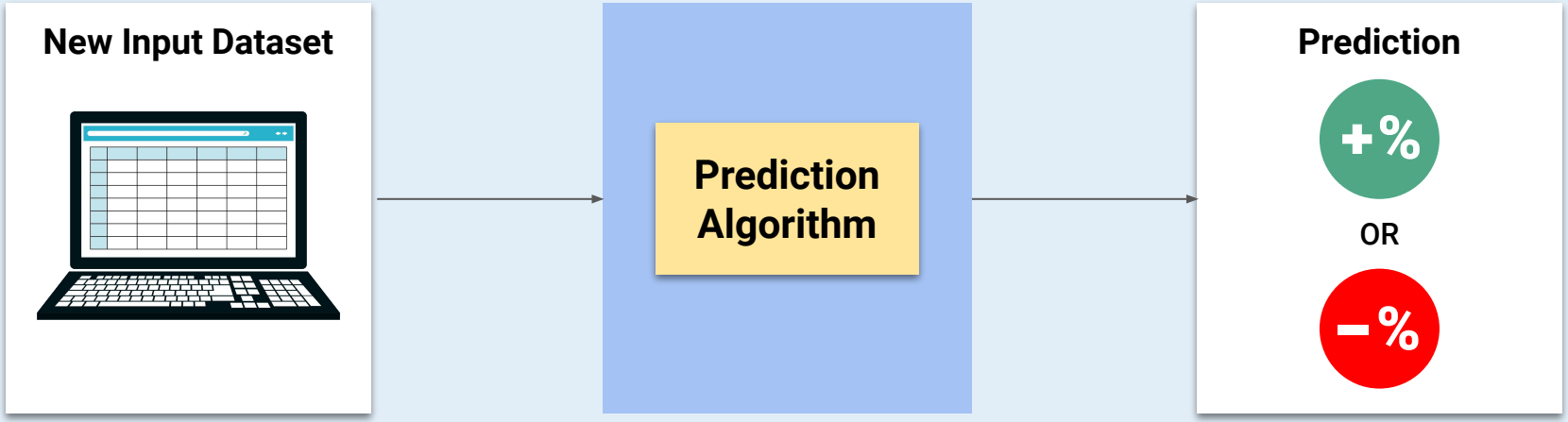
Generating Trading Signal Features

Now that you have learned to generate trading signals, backtest their trading strategies, and evaluate their results, it’s time to incorporate machine learning!

Generating Trading Signal Features

You now have the opportunity to use a machine learning model (Random Forest) to correctly predict next-day positive or negative returns based on multiple trading signals.

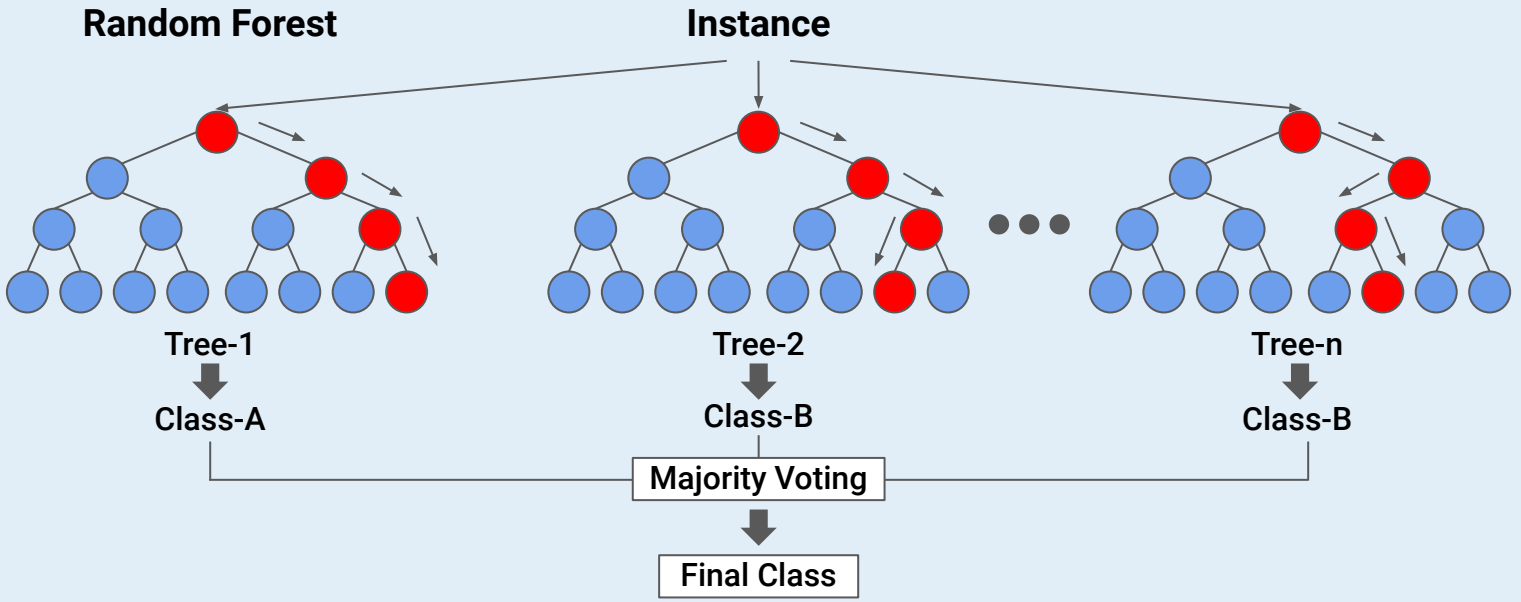
Trained Machine Learning Model



Generating Trading Signal Features

The Random Forest model requires multiple features or, in this case, multiple trading signals to train itself on.

Random Forest Simplified



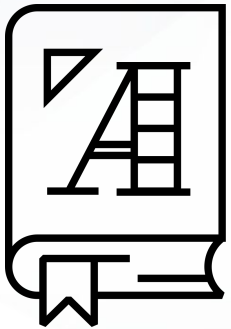
Generating Trading Signal Features

You'll learn to generate multiple trading signals using various technical indicators such as:

01 Exponential moving average of closing prices

02 Exponential moving average of daily return volatility

03 Bollinger Bands, which are a set of lines representing a positive or negative standard deviation away from a simple moving average (SMA) of the asset's closing price



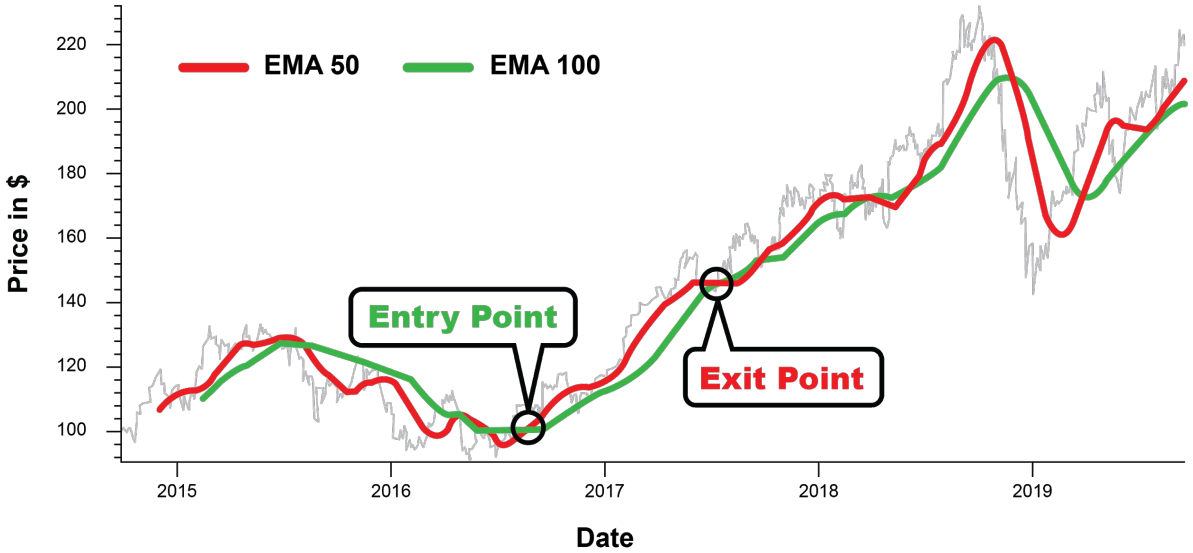
In contrast to a simple moving average (SMA), an **exponential moving average (EMA) of closing prices** represents a moving average with more weight or focus given to the most recent of prices.

Generating Trading Signal Features

A short-window EMA describes “faster” price action than a long-window EMA, its “slower” counterpart.

The logic then dictates:

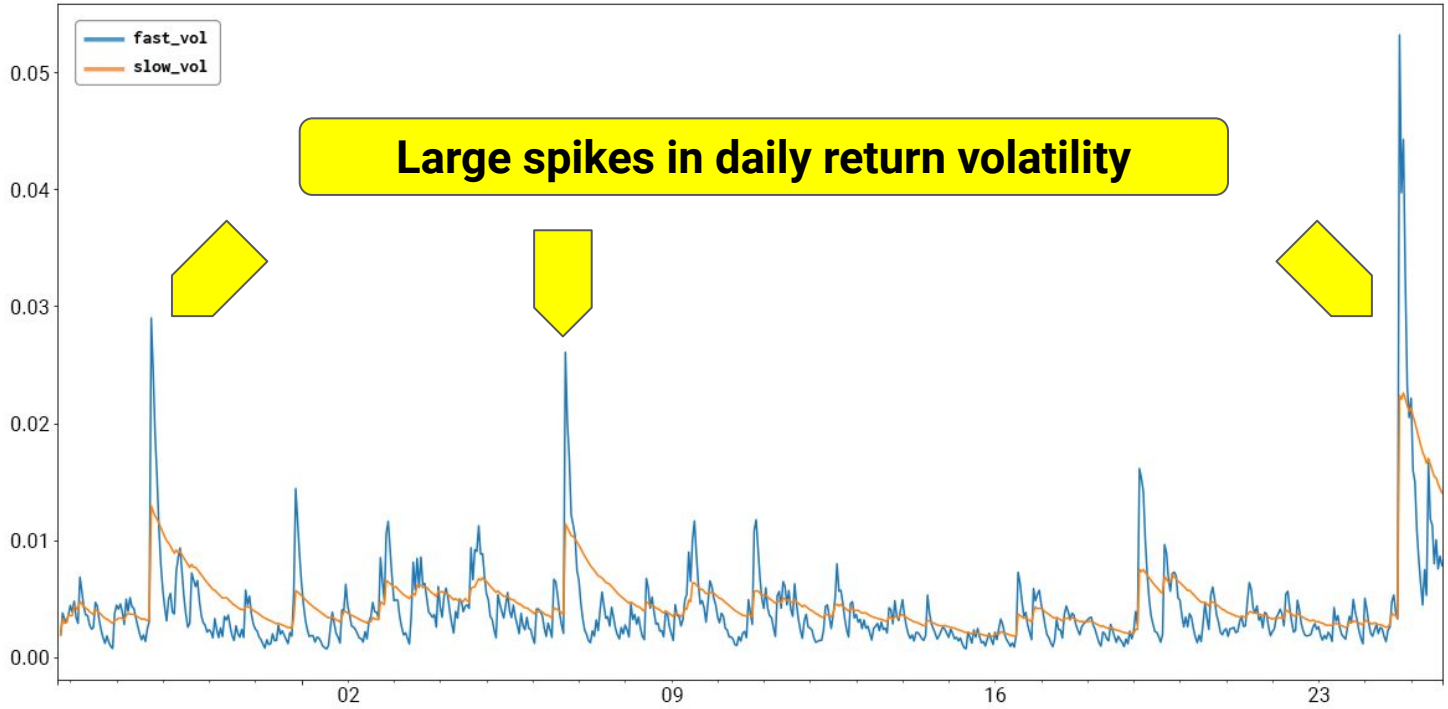
- When the fast EMA is greater than the slow EMA, a long trade opportunity exists, as price action should rise in the short term.
- A short trade opportunity arises for the opposite scenario, in which the slow EMA is greater than the fast EMA.



Similarly, an **exponential moving average of daily return volatility** gives more weight to the most recent of daily returns.

Generating Trading Signal Features

Therefore, when a short-window (fast) EMA of daily return volatility is greater than a long-window (slow) EMA of daily return volatility, the crossover suggests that a short opportunity exists where daily return volatility is expected to rise.



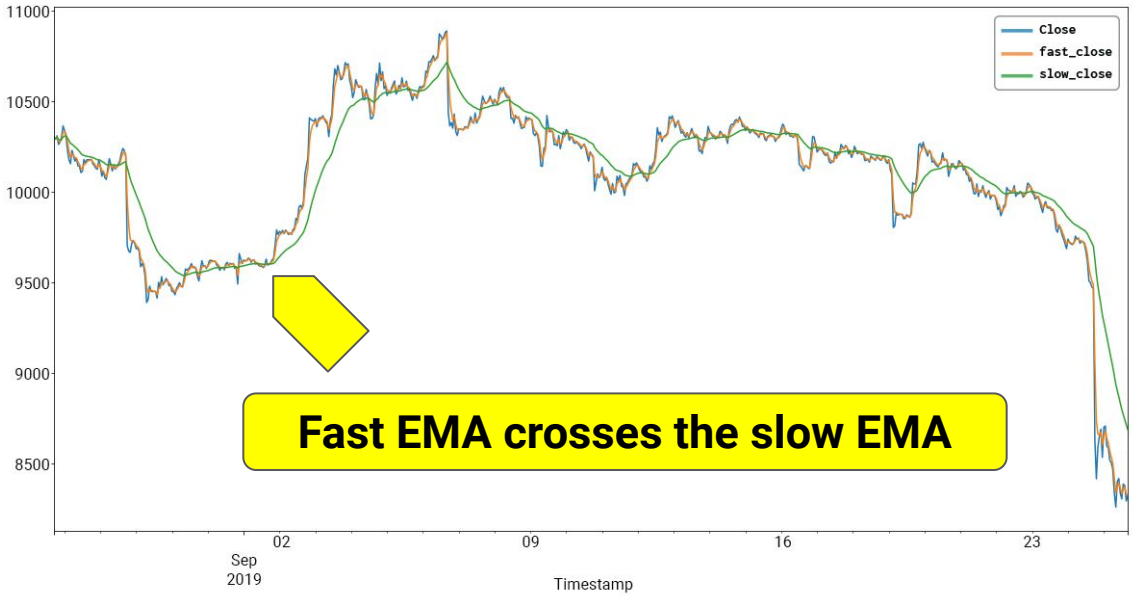
Generating Trading Signal Features

This is because during times of rising price volatility, there often exists a negative price bias (selling), and vice versa for when daily return volatility is expected to fall (buying).

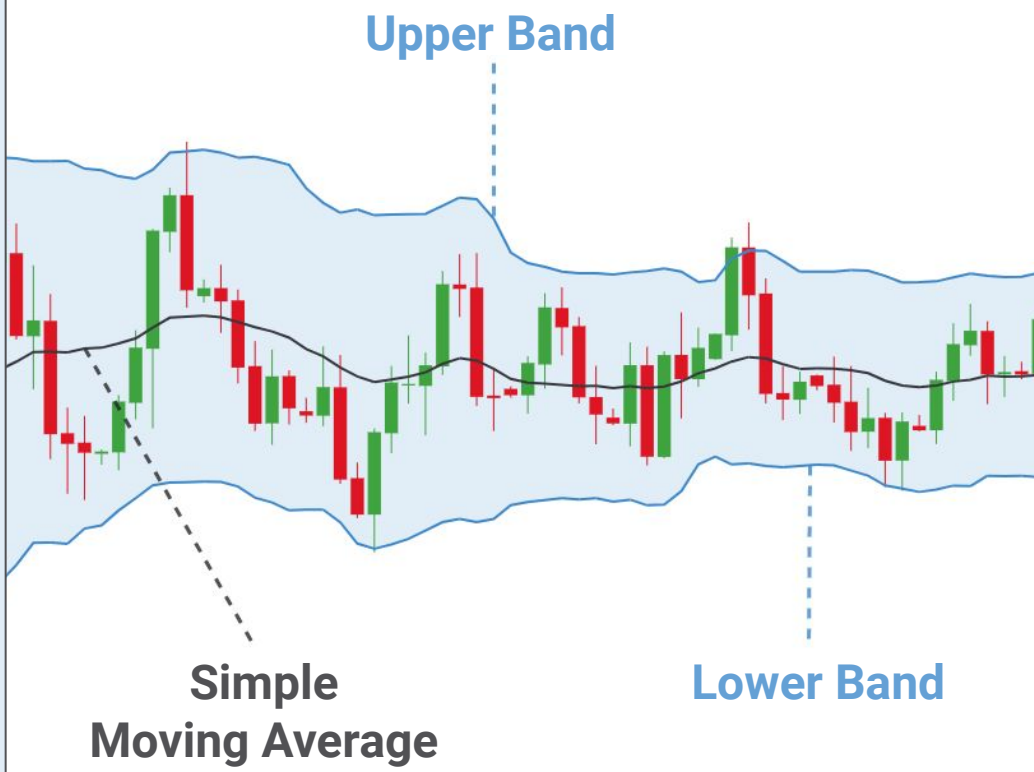
Plot Exponential Moving Average of Closing Prices

```
btc_df[['Close', 'fast_close', 'slow_close']].plot(figsize=(20,10))
```

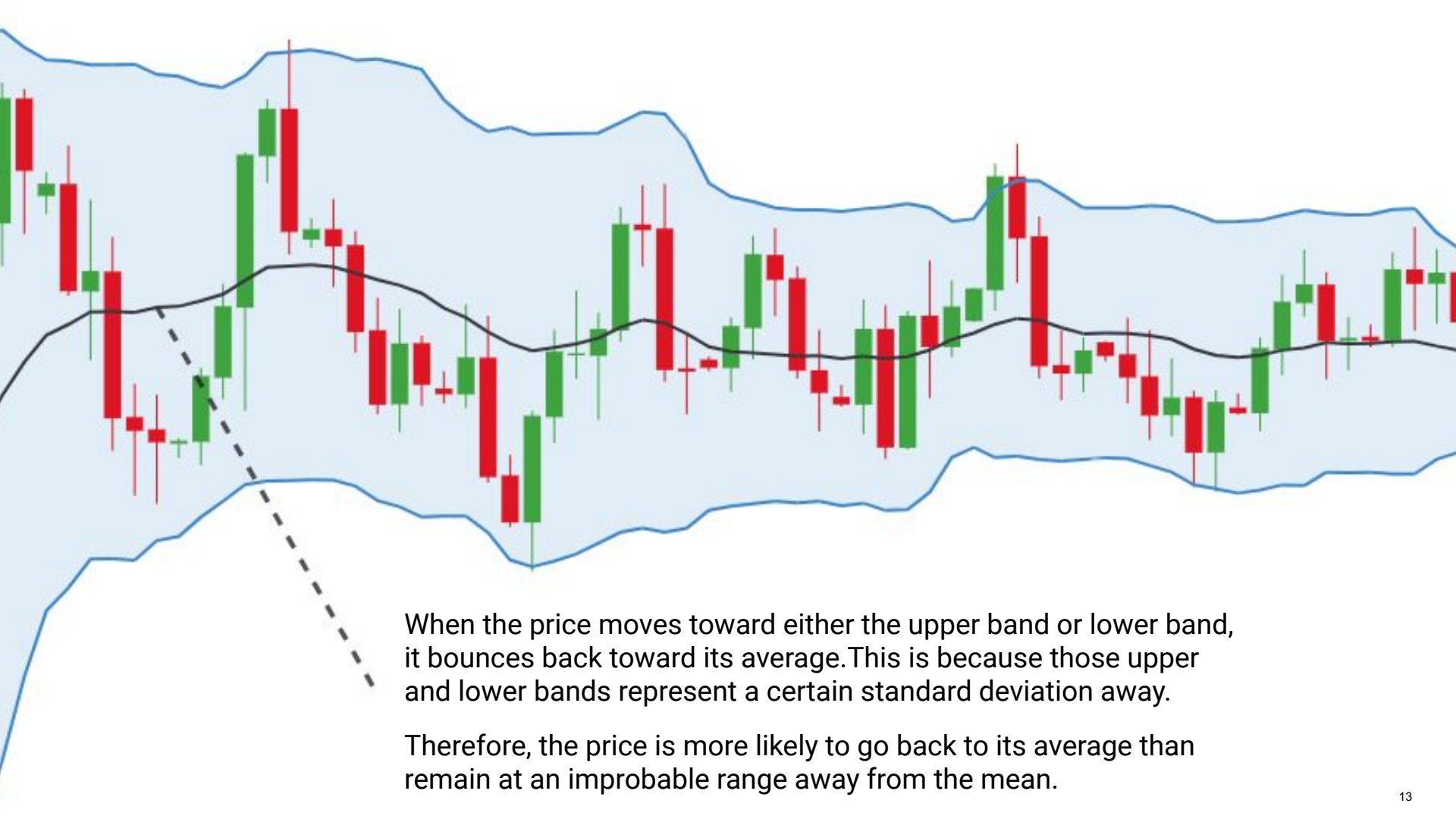
```
<matplotlib.axes._subplots.AxesSubplot at 0x11ed19518>
```



Lastly, a **Bollinger Band** describes a middle, upper, and lower band, in which the middle is a simple moving average (SMA) of closing prices, while the upper and lower bands describe the rolling standard deviation above and below the SMA, respectively.



(dailyfx.com)



When the price moves toward either the upper band or lower band, it bounces back toward its average.This is because those upper and lower bands represent a certain standard deviation away.

Therefore, the price is more likely to go back to its average than remain at an improbable range away from the mean.

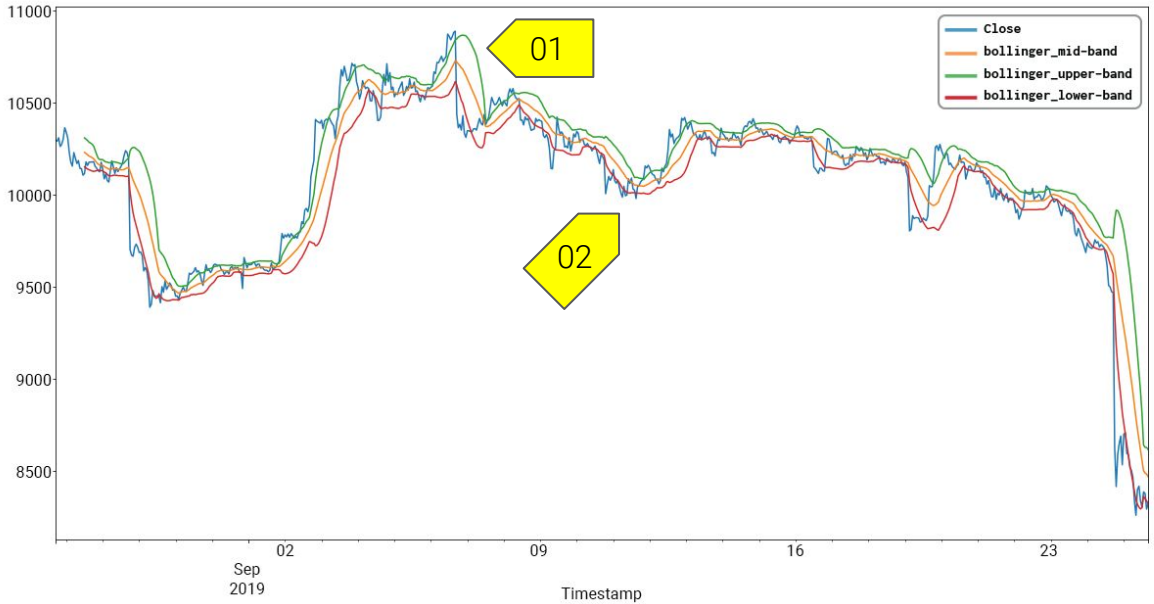
Generating Trading Signal Features

Therefore, when the asset closing price is less than the lower band, a long opportunity exists: the signal suggests that the price action will tend to move upward and more in line with where the price should be (within the negative standard deviation).

Plot Bollinger Bands

```
btc_df[['Close','bollinger_mid_band','bollinger_upper_band','bollinger_lower_band']].plot(figsize=(20,10))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x126db70b8>
```



01 Closing price is above lower band; price moves down.

02 Closing price is below lower band; price moves up.

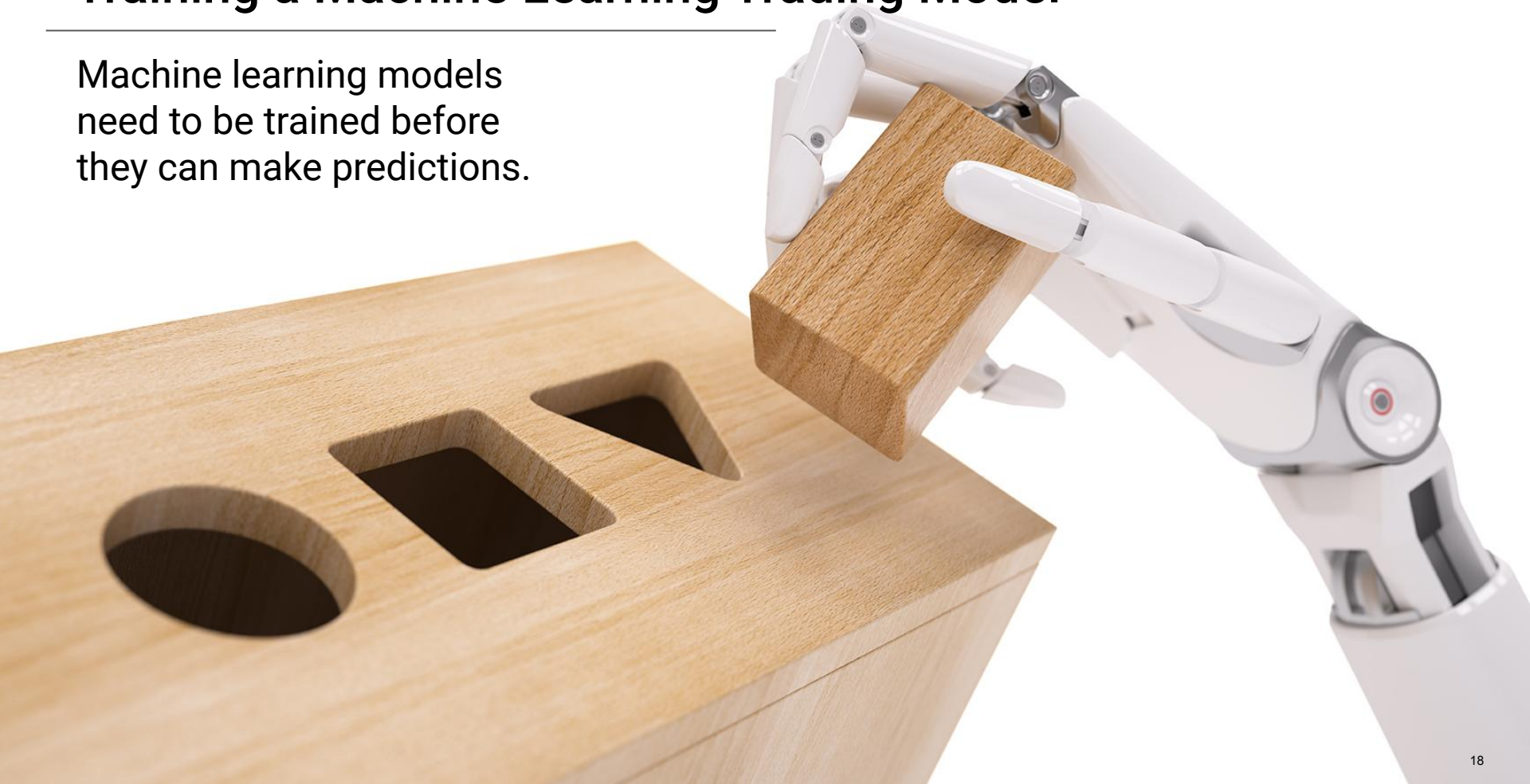
A short opportunity exists for the opposite scenario, in which the asset closing price is greater than the upper band, suggesting that the price action will tend to move downward and within the positive standard deviation.

Training a Machine Learning Trading Model

How can a machine learning model be applied to trading?

Training a Machine Learning Trading Model

Machine learning models need to be trained before they can make predictions.



Training a Machine Learning Trading Model

Therefore, using the trading signals generated from the previous activity, the Random Forest model will train itself using the trading signals as independent variables that determine a dependent variable (a positive or negative return for the next day).

```
# Set path to CSV and read in CSV
csv_path = Path('../Resources/x_test.csv')
x_test=pd.read_csv(csv_path)
x_test.set_index(pd.to_datetime(x_test['Timestamp'], infer_datetime_format=True), inplace=True)
x_test.drop(columns=['Timestamp'], inplace=True)
x_test.head()
```

Timestamp	crossover_signal	vol_trend_signal	bollinger_signal
2019-09-15 00:00:00+00:00	1.0	1.0	0.0
2019-09-15 01:00:00+00:00	1.0	1.0	0.0
2019-09-15 02:00:00+00:00	1.0	1.0	0.0
2019-09-15 03:00:00+00:00	1.0	1.0	0.0
2019-09-15 04:00:00+00:00	-1.0	1.0	0.0

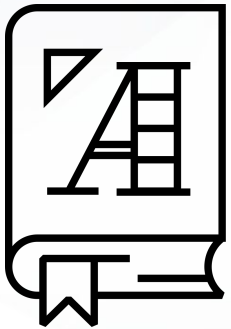
Training a Machine Learning Trading Model

The model can then be saved as a pre-trained model for later use and loaded again for easy deployment.

```
results["Predicted Value"] = predictions
results
```

Timestamp	Return	Actual Value	Predicted Value
2019-09-15 00:00:00+00:00	-0.002268	0	0.0
2019-09-15 01:00:00+00:00	0.001944	1	0.0
2019-09-15 02:00:00+00:00	-0.001602	0	0.0
2019-09-15 03:00:00+00:00	-0.001769	0	0.0
2019-09-15 04:00:00+00:00	0.000920	1	1.0
...	...	...	...
2019-09-25 12:00:00+00:00	-0.003957	0	1.0
2019-09-25 13:00:00+00:00	0.009933	1	1.0
2019-09-25 14:00:00+00:00	-0.001097	0	1.0
2019-09-25 15:00:00+00:00	-0.010085	0	1.0
2019-09-25 16:00:00+00:00	0.003532	1	1.0

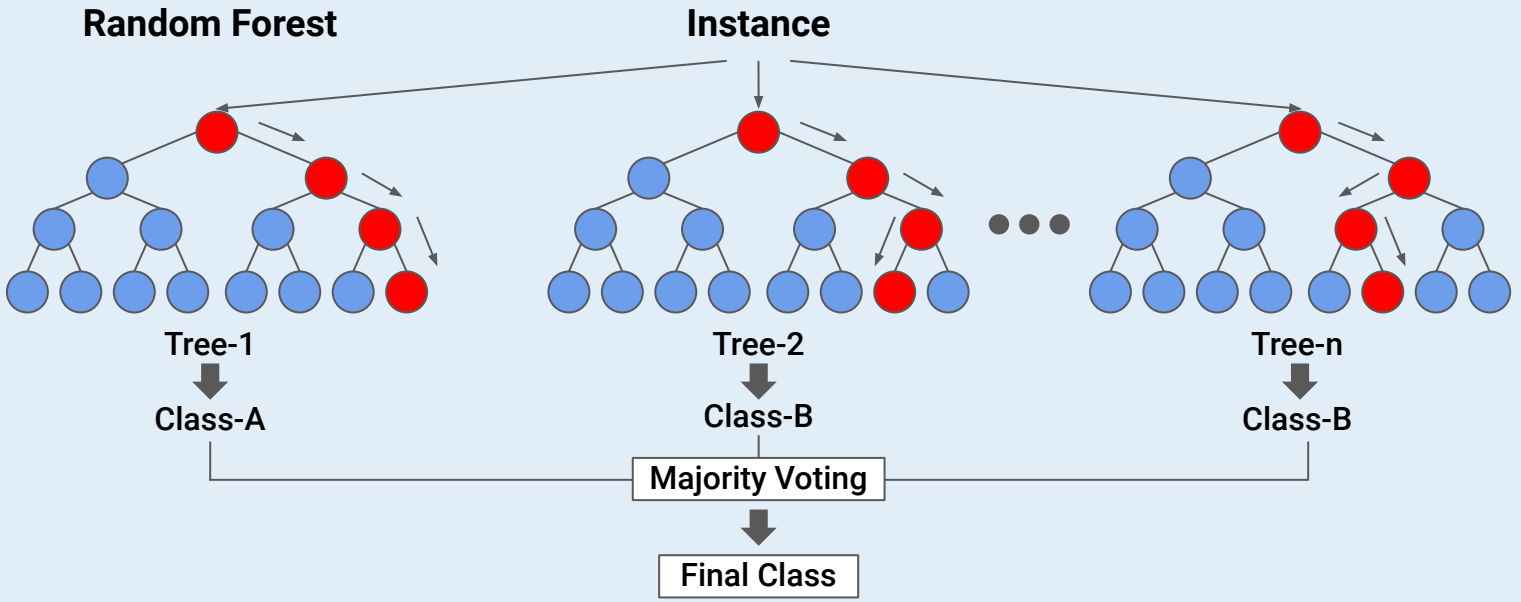
What is a Random Forest model?



A **Random Forest model** is among one of the best-supervised algorithms in terms of its ability to predict outcomes.

Training a Machine Learning Trading Model

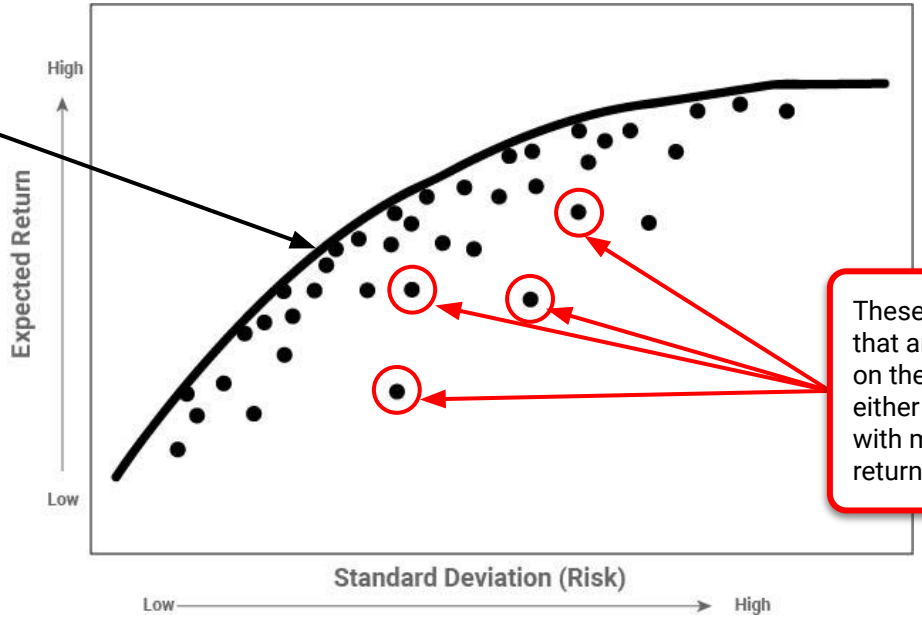
The Random Forest model uses a combination of multiple decision tree models to "average away" or minimize the impact of any single decision tree with high variance. This creates a more reliable predicted result derived from the strongest features.



Training a Machine Learning Trading Model

For example, combining the concepts of Sharpe ratios and portfolio diversification tends to create a portfolio of maximum expected return with minimal variance or risk. This is due to the tendency of the non-correlated stock to “cancel out” each other’s variances.

Each point on this line represents an optimal combination of securities that maximizes the return for any given level of risk (standard deviation).



These dots represent portfolios that are inferior to the portfolios on the efficient frontier—they either offer the same returns but with more risk, or they offer less return for the same risk.

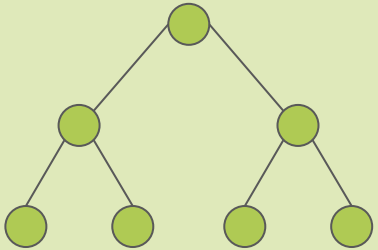
(investinganswers.com)

Why is it called a Random Forest?

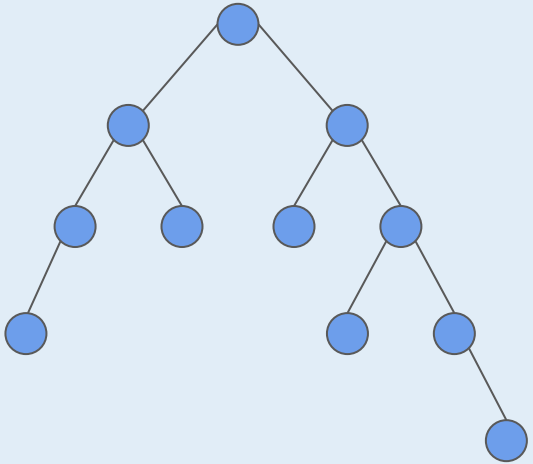
Training a Machine Learning Trading Model

The Random Forest model is a combination of many decision tree models with each decision tree randomly selecting a subset of the observations and features to train itself on. The result is a final prediction that is an average across this “forest” of random trees.

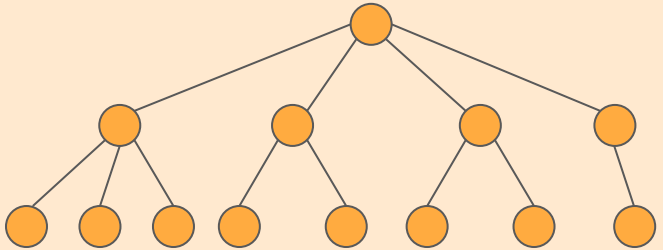
Balanced Tree



Deep Tree



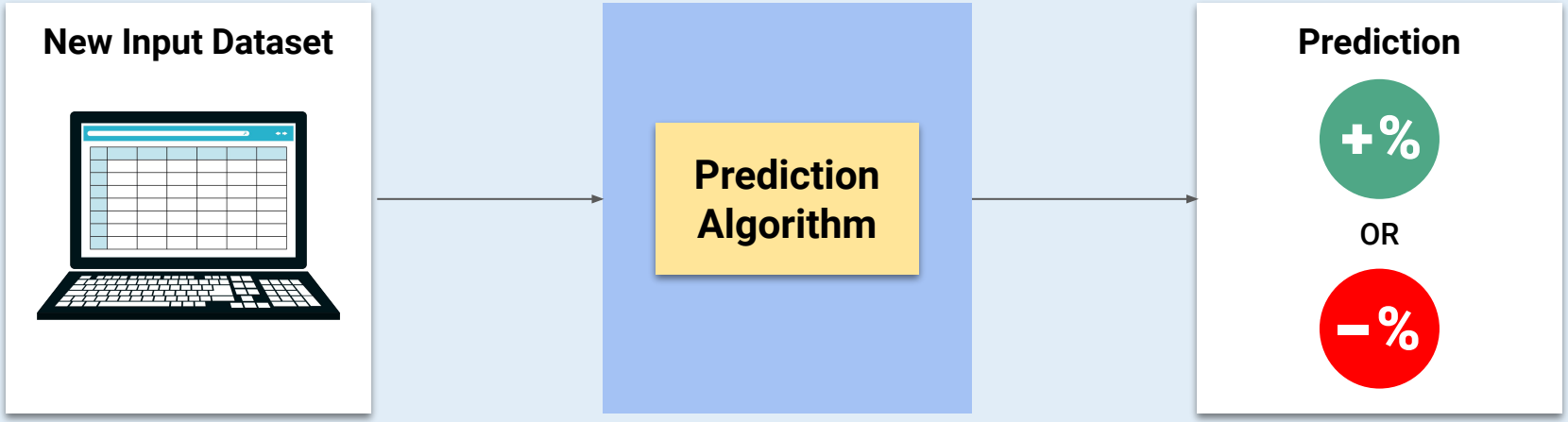
Bushy Tree



Training a Machine Learning Trading Model

Now that the pre-trained Random Forest model has been saved, re-deploying the model to make predictions becomes very straightforward.

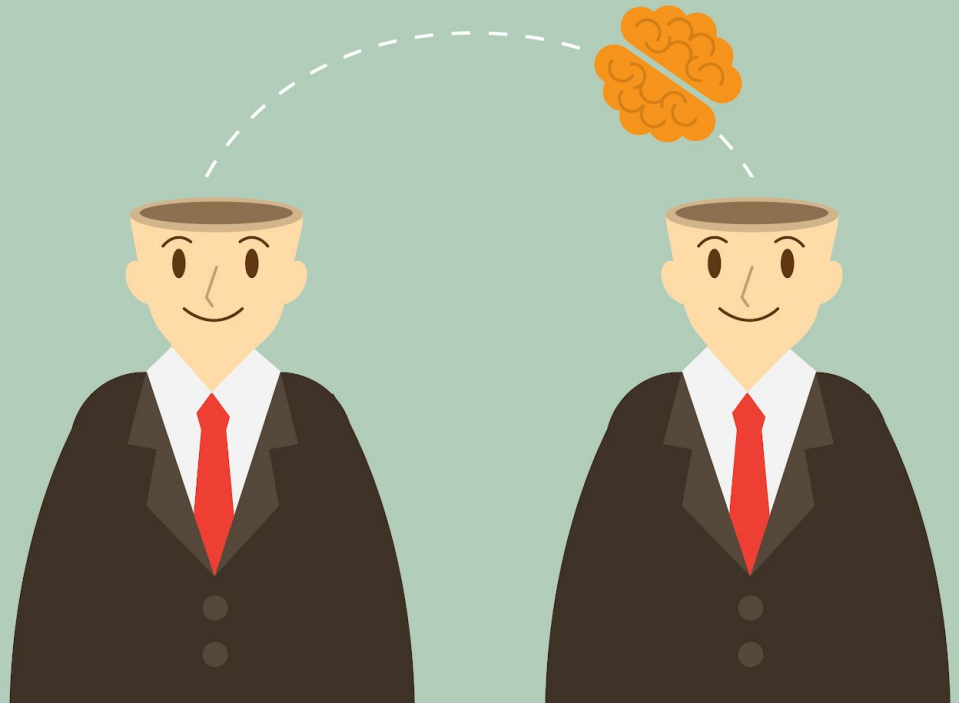
Trained Machine Learning Model



All that needs to be done is to feed in the *x* test data (trading signal data) and compare against the *y* test data (actual daily return results).

Training a Machine Learning Trading Model

Deploying an already trained model saves time and effort, offloading the need for developers to prepare the data, split the data (train and test datasets), and fit the model before finally being able to use the model to make predictions.



Recap

What did we learn today?

Recap

We learned how to implement a machine learning model (Random Forest) to make predictions of next-day daily returns, given a set of trading signals derived from raw asset closing prices.

What was the process for implementing a machine learning trading model?

Recap

The process for implementing a machine learning model, regardless of domain, generally includes the following:

01 Preparing the data

02 Splitting the data into train and test datasets

03 Fitting the *x* and *y* train data to the model

04 Making predictions from the *x* test data

05 Comparing the predicted results to the *y* test data (actual results) to evaluate the performance of the overall model

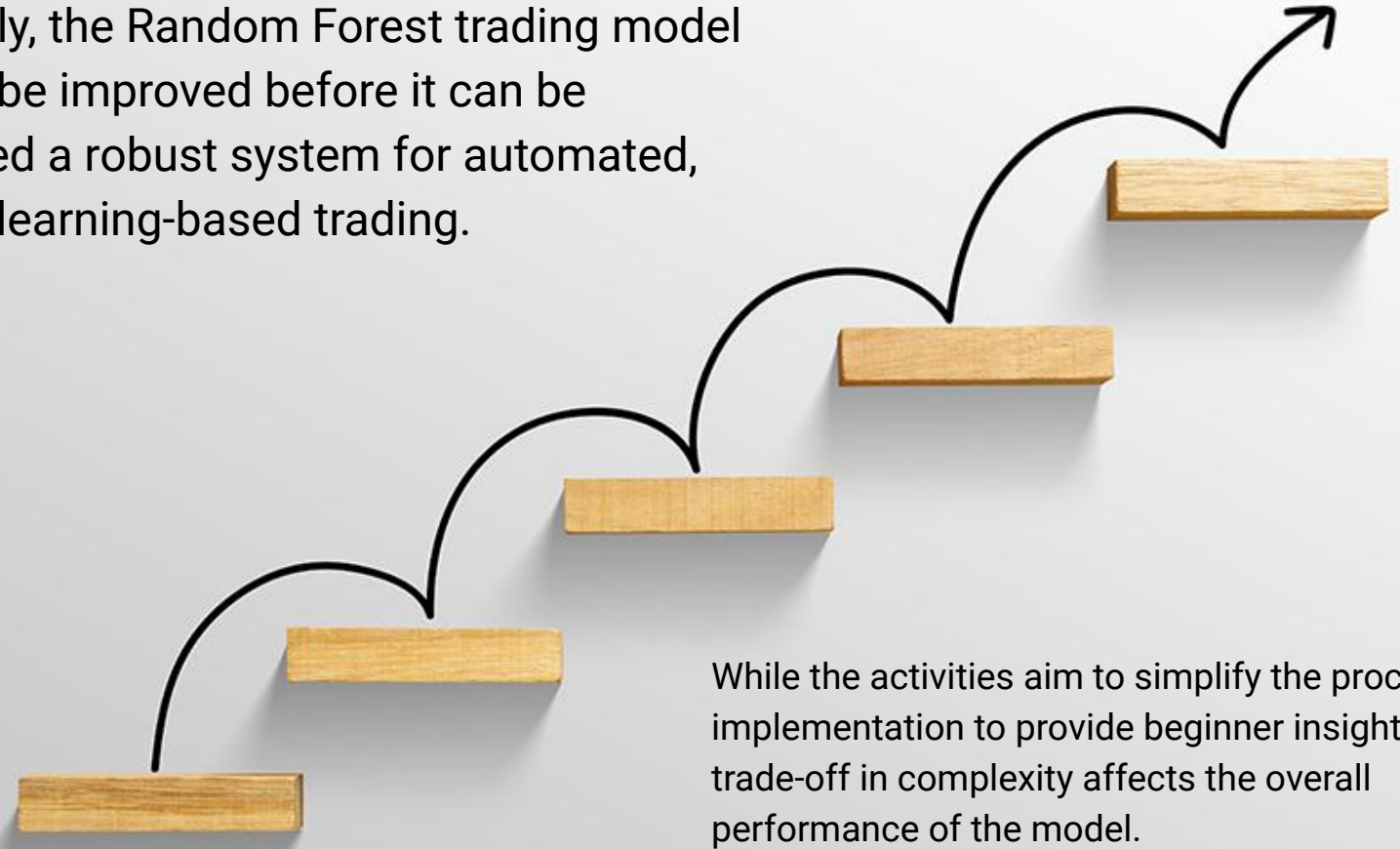
What was the main takeaway of today’s lesson?

Recap

The process for implementing a machine learning trading model can be fairly straightforward.

But the ability to construct a sufficiently sophisticated trading model that can outperform the markets requires more effort; we need to gain a better understanding of the markets and fine-tune the model (add more features and therefore information).

Admittedly, the Random Forest trading model needs to be improved before it can be considered a robust system for automated, machine learning-based trading.



While the activities aim to simplify the process for implementation to provide beginner insight, the trade-off in complexity affects the overall performance of the model.

Recap

Several factors would benefit the training and, therefore, overall performance of the Random Forest trading model, such as:

01

Using more observations or data

02

More features or variables

03

Continuous rather than binary calculations for trading signals

The number of observations and features supplied to the model for training can be increased to provide more information, which would enable the model to make more accurate predictions.

Recap

In this case, there were only 462 observations and 3 features on which the model was trained.

Separate X and Y Training Datasets

```
x_train = trading_signals_df[x_var_list][training_start:training_end]
y_train = trading_signals_df['Positive Return'][training_start:training_end]

x_train.tail()
```

Timestamp	crossover_signal	vol_trend_signal	bollinger_signal
2019-09-14 19:00:00+00:00	1.0	1.0	-1.0
2019-09-14 20:00:00+00:00	1.0	1.0	-1.0
2019-09-14 21:00:00+00:00	1.0	1.0	-1.0
2019-09-14 22:00:00+00:00	1.0	1.0	-1.0
2019-09-14 23:00:00+00:00	1.0	1.0	0.0

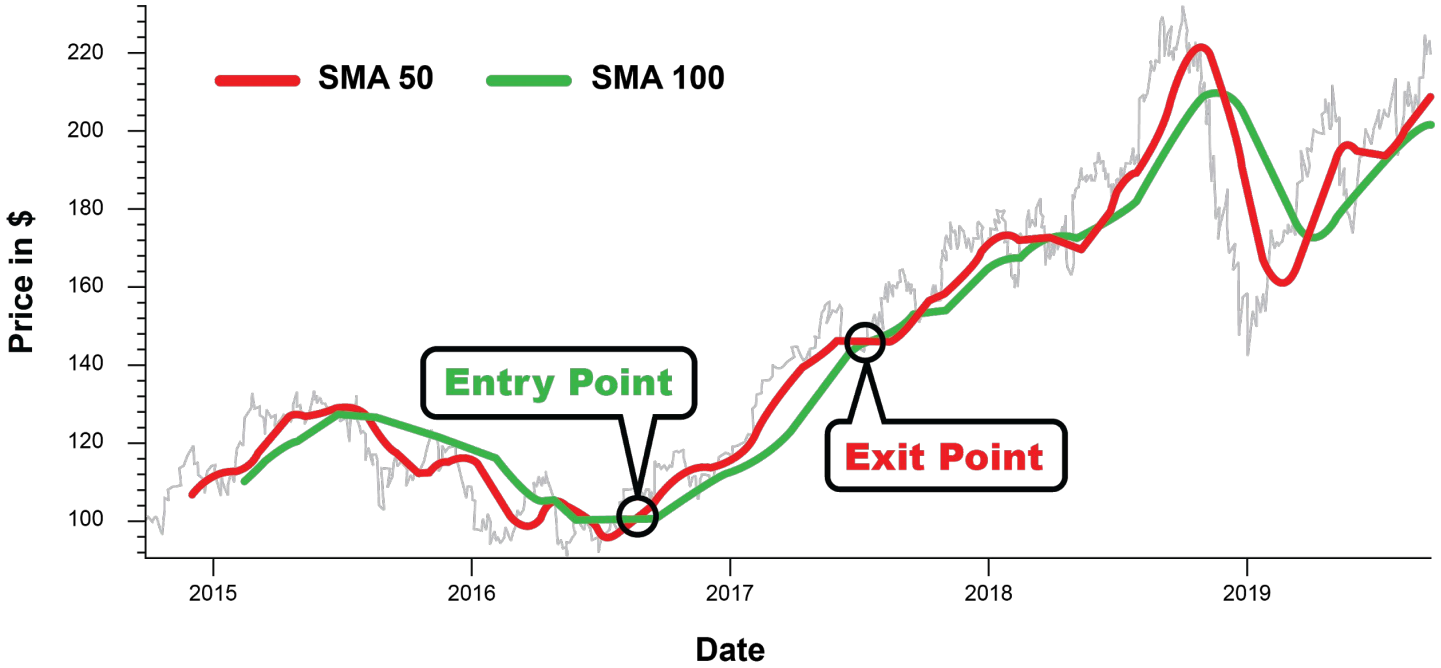
```
x_train.shape
```

```
(462, 3)
```

In addition, for simplicity, the trading signals were output as binary calculation—either ***0*** do not engage in the trade, or ***1*** engages in the trade.

Recap

However, if a scaled continuous value was used instead of a binary value, the extent to which a trading signal is defined or how far the values differ from the crossover point could be used, therefore providing more information to train the model.



Finally, a lot of time and effort is spent collecting and preparing training data.

Recap

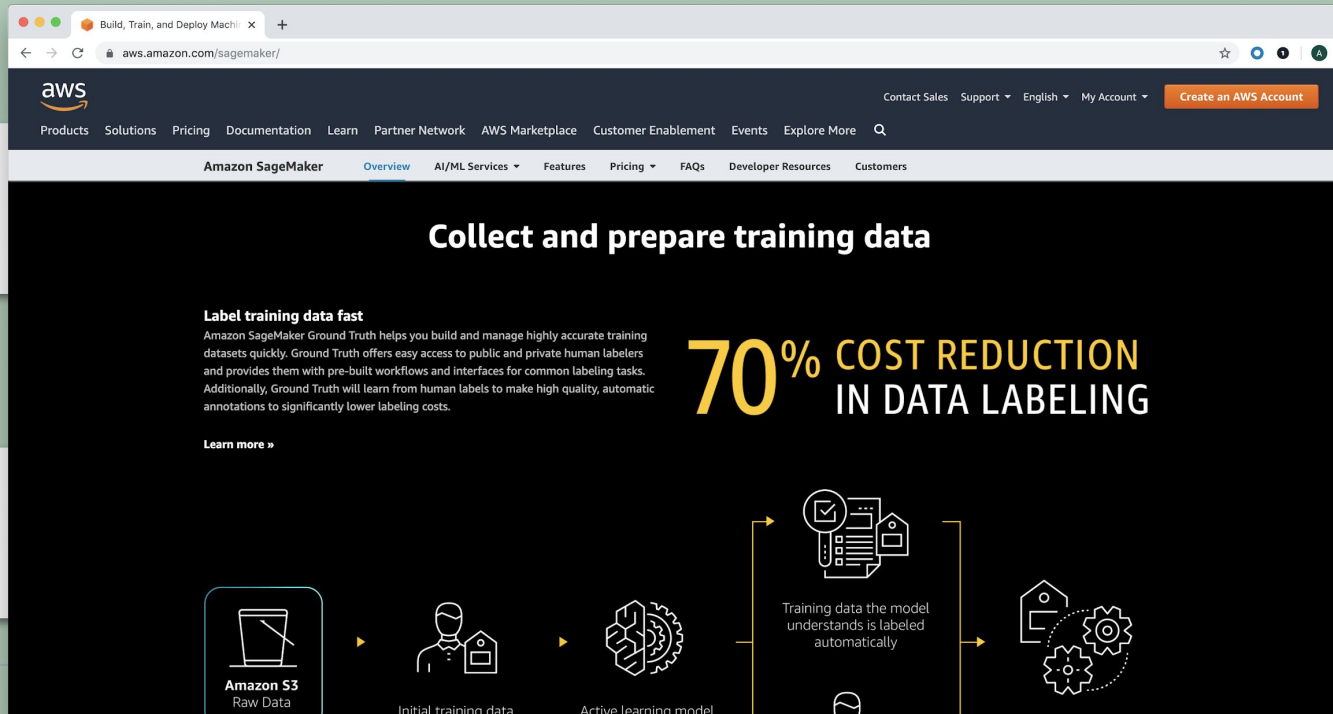
An alternative solution is to use Amazon SageMaker, a machine learning cloud service that enables users to build, train, and deploy machine learning models quickly and conveniently. This would do the following:

01

Minimize the effort spent to prepare data

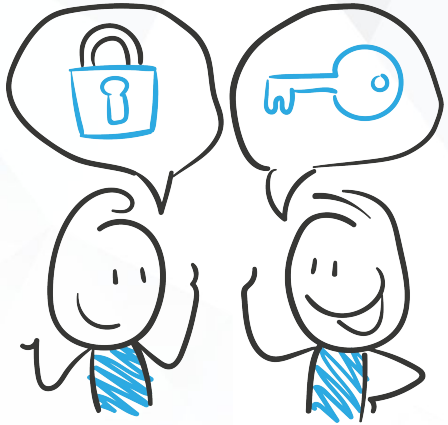
02

Optimize the accuracy or performance of the model



Recap

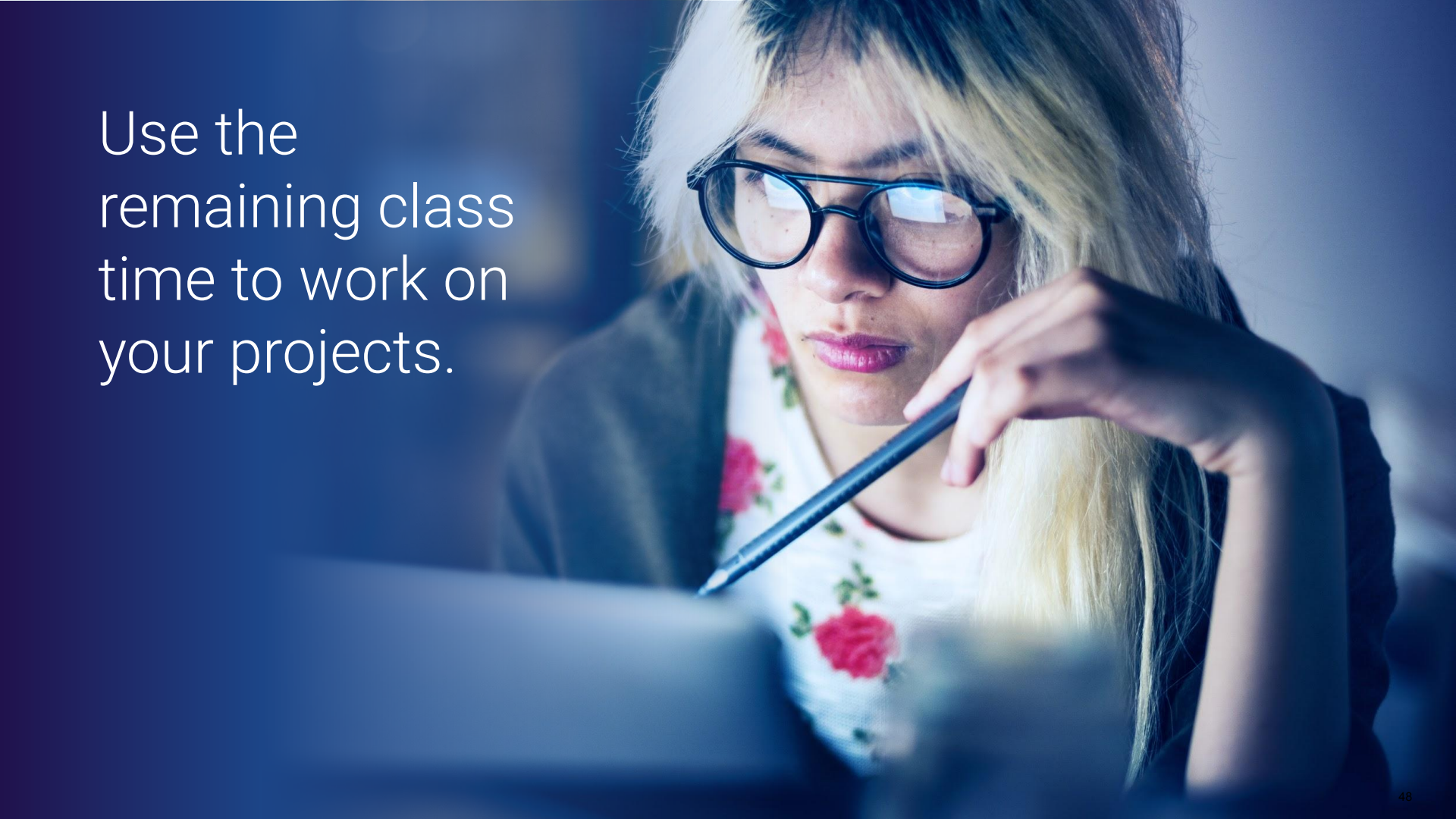
Amazon SageMaker provides several methods for accessing its functionality, such as the AWS web GUI, specific API endpoints, and the SageMaker Python SDK.



Take a moment to reflect on what you just learned.

Questions?

Use the remaining class time to work on your projects.




The End