Alpha Factor Library

Throughout this book, we've described how to engineer features from market, fundamental, and alternative data to build **machine learning** (**ML**) models that yield signals for a trading strategy. The smart design of features, including appropriate preprocessing and denoising, is what typically leads to an effective strategy. This appendix synthesizes some of the lessons learned on feature engineering and provides additional information on this important topic.

Chapter 4, *Financial Feature Engineering – How to Research Alpha Factors*, summarized the long-standing efforts of academics and practitioners to identify information or variables that help reliably predict asset returns. This research led from the single-factor capital asset pricing model to a "zoo of new factors" (Cochrane, 2011). This *factor zoo* contains hundreds of firm characteristics and security price metrics presented as statistically significant predictors of equity returns in the anomalies literature since 1970 (see a summary in Green, Hand, and Zhang, 2017).

Chapter 4, *Financial Feature Engineering – How to Research Alpha Factors*, categorized factors by the underlying risk they represent and for which an investor would earn a reward above and beyond the market return. These categories include value versus growth, quality, and sentiment, as well as volatility, momentum, and liquidity. Throughout this book, we used numerous metrics to capture these risk factors. This appendix expands on those examples and collects popular indicators so you can use it as a reference or inspiration for your own strategy development. It also shows you how to compute them and includes some steps to evaluate these indicators.

To this end, we'll focus on the broad range of indicators implemented by TA-Lib (see *Chapter 4*, *Financial Feature Engineering – How to Research Alpha Factors*) and the *101 Formulaic Alphas* paper (Kakushadze 2016), which presents real-life quantitative trading factors used in production with an average holding period of 0.6-6.4 days. To facilitate replication, we'll limit this review to indicators that rely on readily available market

data. This restriction notwithstanding, the vast and rapidly evolving scope of potentially useful data sources and features implies that this overview is far from comprehensive.

Throughout this chapter, we will use P_t for the closing price and V_t for the trading volume of an asset at time t. Where necessary, superscripts like P_t^{high} or P_t^{H} differentiate between open, high, low, or close prices. r_t denotes the simple return for the period return at time t.

$P_{t-d,t} = \{p_{t-d}, p_{t-d+1}, \ldots, p_t\}$ and $R_{t-d,t} = \{r_{t-d}, r_{t-d+1}, \ldots, r_t\}$ refer to a time series of prices and returns, respectively, from *t-d* to *t*.

Common alpha factors implemented in TA-Lib

The TA-Lib library is widely used to perform technical analysis of financial market data by trading software developers. It includes over 150 popular indicators from multiple categories that range from overlap studies, including moving averages and Bollinger Bands, to statistic functions such as linear regression. The following table summarizes the main categories:

Function Group	# Indicators
Overlap Studies	17
Momentum Indicators	30
Volume Indicators	3
Volatility Indicators	3
Price Transform	4
Cycle Indicators	5
Math Operators	11

Math Transform	15
Statistic Functions	9

There are also over 60 functions that aim to recognize candlestick patterns popular with traders that rely on the visual inspection of charts. Given the mixed evidence on their predictive ability (Horton 2009; Marshall, Young, and Rose 2006), and the goal of learning such patterns from data using the ML algorithms covered in this book, we will focus on the categories listed in the preceding table. Specifically, we will focus on moving averages, overlap studies, momentum, volume and liquidity, volatility, and fundamental risk factors in this section.

See the notebook `common_alpha_factors` for the code examples in this section and additional implementation details regarding TA-Lib indicators. We'll demonstrate how to compute selected indicators for an individual stock, as well as a sample of the 500 most-traded US stocks over the 2007-2016 period (see the notebook `sample_selection` for the preparation of this larger dataset).

A key building block – moving averages

Numerous indicators allow for calculation using different types of **moving averages** (**MAs**). They make different tradeoffs between smoothing a series and reacting to new developments. You can use them as building blocks for your own indicators or modify the behavior of existing indicators by altering the type of MA used in its construction, as we'll demonstrate in the next section. The following table lists the available types of MAs, the TA-Lib function to compute them, and the code you can pass to other indicators to select the given type:

Moving Average	Function	Code
Simple	SMA	0
Exponential	EMA	1
Weighted	WMA	2

Double Exponential	DEMA	3
Triple Exponential	TEMA	4
Triangular	TRIMA	5
Kaufman Adaptive	KAMA	6
MESA Adaptive	MAMA	7

In the remainder of this section, we'll briefly outline their definitions and visualize their different behavior.

Simple moving average

For price series P_t with a window of length N, the **simple moving average (SMA)** at time t weighs each data point within the window equally:

$$\mathrm{SMA}(N)_t = \frac{P_{t-N+1} + P_{t-N+2} + P_{t-N+3} + P_t}{N} = \frac{1}{N}\sum_{i=1}^{N} P_{t-N+i}$$

Exponential moving average

For price series P_t with a window of length N, the **exponential moving average (EMA)** at time t, EMA_t, is recursively defined as the weighted average of the current price and the most recent previous EMA_{t-1}, where the weights α and $1 - \alpha$ are defined as follows:

$$\mathrm{EMA}(N)_t = \alpha P_t + (1 - \alpha)\mathrm{EMA}(N)_{t-1}$$

$$\alpha = \frac{2}{N + 1}$$

Weighted moving average

For price series P_t with a window of length N, the **weighted moving average** (**WMA**) at time t is computed such that the weight of each data point corresponds to its index within the window:

$$\mathrm{WMA}(N)_t = \frac{P_{t-N+1} + 2P_{t-N+2} + 3P_{t-N+3} + NP_t}{N(N+1)/2}$$

Double exponential moving average

The **double exponential moving average** (**DEMA**) for a price series P_t at time t, DEMA_t, is based on the EMA designed to react faster to changes in price. It is computed as the difference between twice the current EMA and the EMA applied to the current EMA, labeled $\mathrm{EMA}_2(N)_t$:

$$\mathrm{DEMA}(N)_t = 2 \times \mathrm{EMA}(N)_t - \mathrm{EMA}_2(N)_t$$

Since the calculation uses EMA_2, DEMA needs $2 \times N - 1$ samples to start producing values.

Triple exponential moving average

The **triple exponential moving average** (**TEMA**) for a price series P_t at time t, TEMA_t, is also based on the EMA, yet designed to react even faster to changes in price and indicate short-term price direction. It is computed as the difference between three times the difference between the current EMA and the EMA applied to the current EMA, EMA_2, with the addition of the EMA applied to the EMA_2, labeled EMA_3:

$$\mathrm{TEMA}(N)_t = 3 \times [\mathrm{EMA}(N)_t - \mathrm{EMA}_2(N)_t] + \mathrm{EMA}_3(N)_t$$

Since the calculation uses EMA_3, DEMA needs $3 \times N - 2$ samples to start producing values.

Triangular moving average

The **triangular moving average** (**TRIMA**) with window length N for a price series P_t at time t, $\mathrm{TRIMA}(N)_t$, is a weighted average of the last N

SMA(N)$_t$ values. In other words, it applies the SMA to a time series of SMA values:

$$\text{TRIMA}(N)_t = \frac{1}{N}\sum_{i=1}^{N} \text{SMA}(N)_{t-N+i}$$

Kaufman adaptive moving average

The computation of the **Kaufman adaptive moving average** (**KAMA**) aims to take into account changes in market volatility. See the notebook for links to resources that explain the details of this slightly more involved computation.

MESA adaptive moving average

The **MESA adaptive moving average** (**MAMA**) is an exponential moving average that adapts to price movement based on the rate change of phase, as measured by the **Hilbert Transform discriminator** (see TA-Lib documentation). In addition to the price series, MAMA accepts two additional parameters, *fastlimit* and *slowlimit*, that control the maximum and minimum alpha values that should be applied to the EMA when calculating MAMA.

Visual comparison of moving averages

Figure A.1 illustrates how the behavior of the different MAs differs in terms of smoothing the time series and adapting to recent changes. All the time series are calculated for a 21-day moving window (see the notebook for details and color images):

Figure A.1: Comparison of MAs for AAPL closing price

Overlap studies – price and volatility trends

TA-Lib includes several indicators aimed at capturing recent trends, as listed in the following table:

Function	Name
BBANDS	Bollinger Bands
HT_TRENDLINE	Hilbert Transform – Instantaneous Trendline
MAVP	Moving average with variable period
MA	Moving average
SAR	Parabolic SAR
SAREXT	Parabolic SAR – Extended

The `MA` and `MAVP` functions are wrappers for the various MAs described in the previous section. We will highlight a few examples in this section;

see the notebook for additional information and visualizations.

Bollinger Bands

Bollinger Bands combine an MA with an upper band and a lower band representing the moving standard deviation. We can obtain the three time series by providing an input price series, the length of the moving window, the multiplier for the upper and lower bands, and the type of MA, as follows:

```
s = talib.BBANDS(df.close,      # No. of periods (2 to 100000)
                 timeperiod=20,
                 nbdevup=2,     # Deviation multiplier for lower band
                 nbdevdn=2,     # Deviation multiplier for upper band
                 matype=1)      # default: SMA
```

For a sample of AAPL closing prices for 2012, we can plot the result like so:

```
bb_bands = ['upper', 'middle', 'lower']
df = price_sample.loc['2012', ['close']]
df = df.assign(**dict(zip(bb_bands, s)))
ax = df.loc[:, ['close'] + bb_bands].plot(figsize=(16, 5), lw=1);
```

The preceding code results in the following plot:

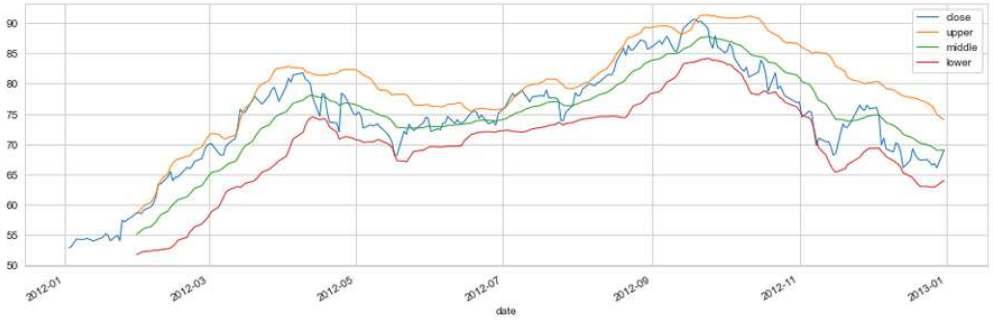

Figure A.2: Bollinger Bands for AAPL close price in 2012

John Bollinger, who invented the concept, also defined over 20 trading rules based on the relationships between the three lines and the current price (see *Chapter 4, Financial Feature Engineering – How to Research*

Alpha Factors). For example, a smaller distance between the outer bands implies reduced recent price volatility, which, in turn, is interpreted as greater volatility and price change going forward.

We can standardize the security-specific values of the Bollinger Bands by forming ratios between the upper and lower bands, as well as between each of them and the close price, as follows:

```
fig, ax = plt.subplots(figsize=(16,5))
df.upper.div(df.close).plot(ax=ax, label='bb_up')
df.lower.div(df.close).plot(ax=ax, label='bb_low')
df.upper.div(df.lower).plot(ax=ax, label='bb_squeeze')
plt.legend()
fig.tight_layout();
```

The following plot displays the resulting normalized time series:

Figure A.3: Normalized Bollinger Band indicators

The following function can be used with the pandas `.groupby()` and `.apply()` methods to compute the indicators for a larger sample of 500 stocks, as shown here:

```
def compute_bb_indicators(close, timeperiod=20, matype=0):
    high, mid, low = talib.BBANDS(close,
                                  timeperiod=20,
                                  matype=matype)
    bb_up = high / close -1
    bb_low = low / close -1
    squeeze = (high - low) / close
    return pd.DataFrame({'BB_UP': bb_up,
                         'BB_LOW': bb_low,
                         'BB_SQUEEZE': squeeze},
```

```
                           index=close.index)
    data = (data.join(data
                      .groupby(level='ticker')
                      .close
                      .apply(compute_bb_indicators)))
```

Figure A.4 plots the distribution of values for each indicator across the 500 stocks (clipped at the 1st and 99th percentiles, hence the spikes in the plots):

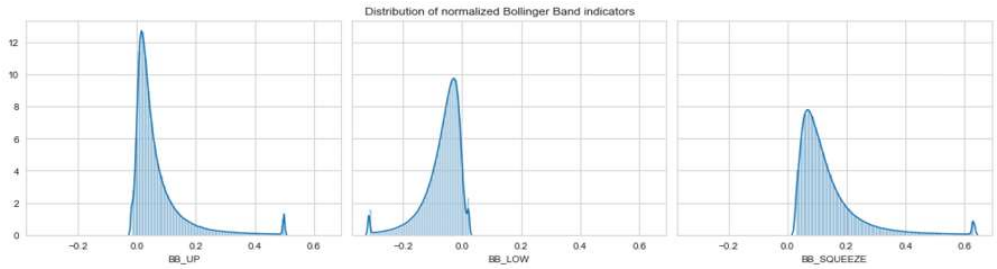

Figure A.4: Distribution of normalized Bollinger Band indicators

Parabolic SAR

The **parabolic SAR** aims to identify trend reversals. It is a trend-following (lagging) indicator that can be used to set a trailing stop loss or determine entry or exit points. It is usually represented in a price chart as a set of dots near the price bars. Generally, when these dots are above the price, it signals a downward trend; it signals an upward trend when the dots are below the price. The change in the direction of the dots can be interpreted as a trade signal. However, the indicator is less reliable in a flat or range-bound market. It is computed as follows:

$$\mathrm{SAR}_t = \mathrm{SAR}_{t-1} + \alpha(\mathrm{EP} - \mathrm{SAR}_{t-1})$$

The **extreme point** (**EP**) is a record that's kept during each trend that represents the highest value reached by the price during the current uptrend —or lowest value during a downtrend. During each period, if a new maximum (or minimum) is observed, the EP is updated with that value.

The α value represents the acceleration factor and is typically set initially to a value of 0.02. This factor increases by α each time a new EP is recorded. The rate will then quicken to a point where the SAR converges toward the price. To prevent it from getting too large, a maximum value for the acceleration factor is normally set to 0.20.

We can compute and plot it for our sample close price series as follows:

```
df = price_sample.loc['2012', ['close', 'high', 'low']]
df['SAR'] = talib.SAR(df.high, df.low,
                      acceleration=0.02, # common value
                      maximum=0.2)
df[['close', 'SAR']].plot(figsize=(16, 4), style=['-', '--']);
```

The preceding code produces the following plot:

Figure A.5: Parabolic SAR for AAPL stock price

Momentum indicators

Chapter 4, *Financial Feature Engineering – How to Research Alpha Factors*, introduced **momentum** as one of the best-performing risk factors historically and listed several indicators designed to identify the corresponding price trends. These indicators include the **relative strength index** (**RSI**), as well as **price momentum** and **price acceleration**:

Factor	Description	Calculation
Relative strength index (RSI)	RSI compares the magnitude of recent price changes across stocks to identify stocks as overbought or oversold. A high RSI (usually above 70) indicates	$\text{RSI} = 100 - \frac{100}{1 + \frac{\Delta_p^{up}}{\Delta_p^{down}}}$

	overbought and a low RSI (typically below 30) indicates oversold. It first computes the average price change for a given number (often 14) of prior trading days with rising ( ) and falling prices ( ), respectively.
Price momentum	This factor computes the total return for a given number of prior trading days d. In the academic literature, it is common to use the last 12 months except for the most recent month due to a short-term reversal effect frequently observed. However, shorter periods have also been widely used.
Price acceleration	Price acceleration calculates the gradient of the price trend using the linear regression coefficient of a time trend on daily prices for a longer and a shorter period, for example, 1 year and 3 months of trading days, and compares the change in the slope as a measure of price acceleration.

TA-Lib implements 30 momentum indicators; the most important ones are listed in the following table. We will introduce a few selected examples; please see the notebook `common_alpha_factors` for additional information:

Function	Name
PLUS_DM/MINUS_DM	Plus/Minus Directional Movement
PLUS_DI/MINUS_DI	Plus/Minus Directional Indicator

DX	Directional Movement Index
ADX	Average Directional Movement Index
ADXR	Average Directional Movement Index Rating
APO/PPO	Absolute/Percentage Price Oscillator
AROON/AROONOSC	Aroon/Aroon Oscillator
BOP	Balance of Power
CCI	Commodity Channel Index
CMO	Chande Momentum Oscillator
MACD	Moving Average Convergence/Divergence
MFI	Money Flow Index
MOM	Momentum
RSI	Relative Strength Index
STOCH	Stochastic
ULTOSC	Ultimate Oscillator
WILLR	Williams' %R

Several of these indicators are closely related and build on each other, as the following example demonstrates.

Average directional movement indicators

The **average directional movement index** (**ADX**) combines two other indicators, namely the positive and negative directional indicators (`PLUS_DI` and `MINUS_DI`), which, in turn, build on the positive and nega-

tive directional movement (`PLUS_DM` and `MINUS_DM`). See the notebook for additional details.

Plus/minus directional movement

For a price series P_t with daily highs P_t^H and daily lows P_t^L, the directional movement tracks the absolute size of price moves over a time period T, as follows:

$$\text{Up}_t = P_t^H - P_{t-T}^H$$

$$\text{Down}_t = P_{t-T}^L - P_t^L$$

$$\text{PLUS\_DM}_t = \begin{cases} \text{Up}_t & \text{if } \text{Up}_t > \text{Down}_t \text{ and } \text{Up}_t > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{MINUS\_DM}_t = \begin{cases} \text{Down}_t & \text{if } \text{Down}_t > \text{Up}_t \text{ and } \text{Down}_t < 0 \\ 0 & \text{otherwise} \end{cases}$$

We can compute and plot this indicator for a 2-year price series of the AAPL stock in 2012-13:

```
df = price_sample.loc['2012': '2013', ['high', 'low', 'close']]
df['PLUS_DM'] = talib.PLUS_DM(df.high, df.low, timeperiod=10)
df['MINUS_DM'] = talib.MINUS_DM(df.high, df.low, timeperiod=10)
```

The following plot visualizes the resulting time series:

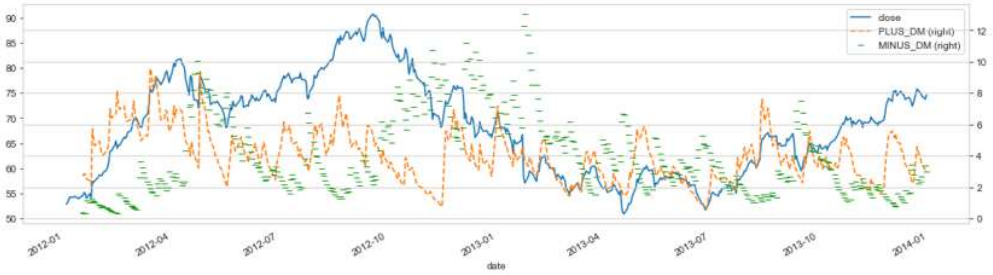

Figure A.6: PLUS_DM/MINUS_DM for AAPL stock price

Plus/minus directional index

`PLUS_DI` and `MINUS_DI` are the simple MAs of `PLUS_DM` and `MINUS_DM`, respectively, each divided by the **average true range** (**ATR**). See the *Volatility indicators* section later in this chapter for more details.

The simple MA is calculated over the given number of periods. The ATR is a smoothed average of the true ranges.

Average directional index

Finally, the **average directional index** (**ADX**) is the (simple) MA of the absolute value of the difference between `PLUS_DI` and `MINUS_DI`, divided by their sum:

$$\text{ADX} = 100 \times \text{SMA}(N)_t \left| \frac{\text{PLUS}_\text{D} I_t - \text{MINUS}_\text{D} \text{I}_t}{\text{PLUS}_\text{D} I_t + \text{MINUS}_\text{D} \text{I}_t} \right|$$

Its values oscillate in the 0-100 range and are often interpreted as follows:

ADX Value	Trend Strength
0-25	Absent or weak trend
25-50	Strong trend
50-75	Very strong trend
75-100	Extremely strong trend

We compute the ADX time series for our AAPL sample series similar to the previous examples, as follows:

```
df['ADX'] = talib.ADX(df.high,
                      df.low,
```

```
                    df.close,
                    timeperiod=14)
```

The following plot visualizes the result over the 2007-2016 period:

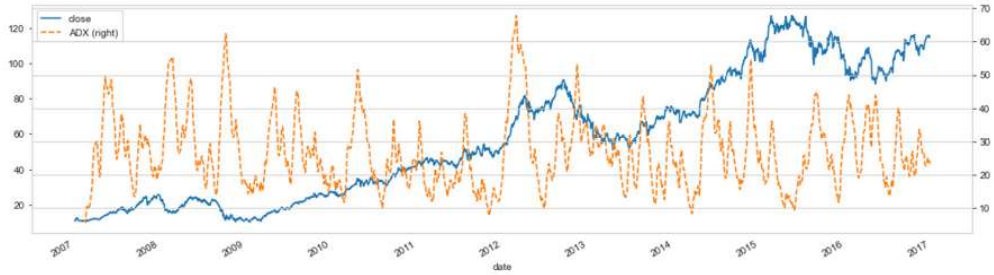

Figure A.7: ADX for the AAPL stock price series

Aroon Oscillator

The Aroon indicator measures the time between highs and the time between lows over a time period. It computes an `AROON_UP` and an `AROON_DWN` indicator, as follows:

$$\text{AROON\_UP} = \frac{T - \text{Periods since T period High}}{T} \times 100$$

$$\text{AROON\_DOWN} = \frac{T - \text{Periods since T period Low}}{T} \times 100$$

The Aroon Oscillator is simply the difference between the `AROON_UP` and `AROON_DOWN` indicators and moves within the range from -100 to 100, as shown here for the AAPL price series:

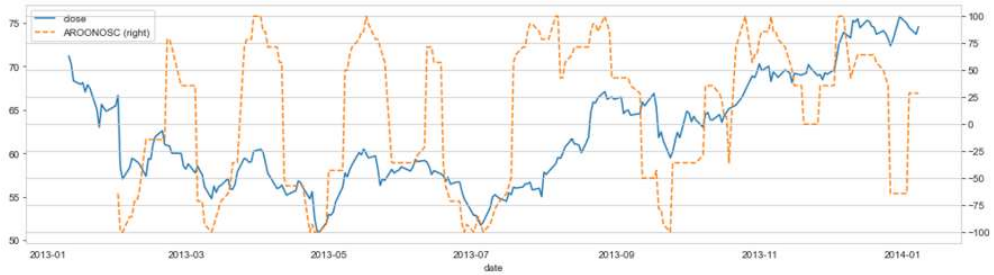

Figure A.8: Aroon Oscillator for the AAPL stock price series

Balance of power

The **balance of power** (**BOP**) intends to measure the strength of buyers relative to sellers in the market by assessing the influence of each side on the price. It is computed as the difference between the close and the open price, divided by the difference between the high and the low price:

$$\text{BOP}_t = \frac{P_t^{\text{Close}} - P_t^{\text{Open}}}{P_t^{\text{High}} - P_t^{\text{Low}}}$$

Commodity channel index

The **commodity channel index** (**CCI**) measures the difference between the current *typical* price, computed as the average of current low, high, and close price and the historical average price. A positive (negative) CCI indicates that the price is above (below) the historic average. It is computed as follows:

$$\bar{P}_t = \frac{P_t^H + P_t^L + P_t^C}{3}$$

$$\text{CCI}_t = \frac{\bar{P}_t - \text{SMA}(N)_t}{0.15 \sum_{t=i}^{T} |\bar{P}_t - \text{SMA}(N)_t| / T}$$

Moving average convergence divergence

Moving average convergence divergence (**MACD**) is a very popular trend-following (lagging) momentum indicator that shows the relation-

ship between two MAs of a security's price. It is calculated by subtracting the 26-period EMA from the 12-period EMA.

The TA-Lib implementation returns the MACD value and its signal line, which is the 9-day EMA of the MACD. In addition, the MACD-Histogram measures the distance between the indicator and its signal line. The following charts show the results:

Figure A.9: The three MACD series for the AAPL stock price series

Stochastic relative strength index

The **stochastic relative strength index (StochRSI)** is based on the RSI described at the beginning of this section and intends to identify crossovers, as well as overbought and oversold conditions. It compares the distance of the current RSI to the lowest RSI over a given time period T to the maximum range of values the RSI has assumed for this period. It is computed as follows:

$$\text{STOCHRSI}_t = \frac{\text{RSI}_t - \text{RSI}_t^L(T)}{\text{RSI}_t^H(T) - \text{RSI}_t^L(T)}$$

The TA-Lib implementation offers more flexibility than the original unsmoothed stochastic RSI version by Chande and Kroll (1993). To calculate

the original indicator, keep `timeperiod` and `fastk_period` equal.

The return value `fastk` is the unsmoothed RSI. `fastd_period` is used to compute a smoothed StochRSI, which is returned as `fastd`. If you do not care about StochRSI smoothing, just set `fastd_period` to 1 and ignore the `fasytd` output:

```
fastk, fastd = talib.STOCHRSI(df.close,
                              timeperiod=14,
                              fastk_period=14,
                              fastd_period=3,
                              fastd_matype=0)
df['fastk'] = fastk
df['fastd'] = fastd
```

Figure A.10 plots the closing price and both the smoothed and unsmoothed stochastic RSI:

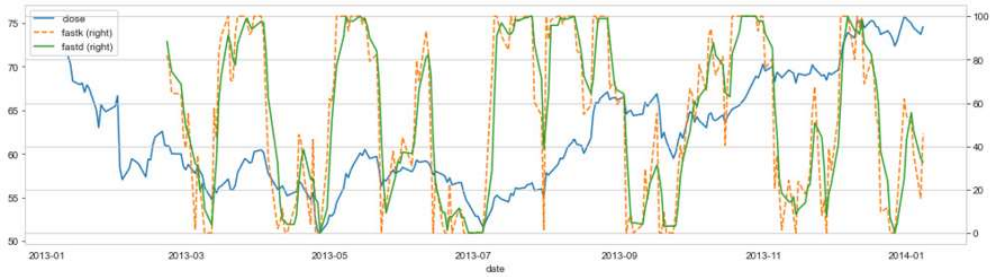

Figure A.10: Smoothed and unsmoothed StochRSI series for the AAPL stock price

Stochastic oscillator

A stochastic oscillator is a momentum indicator that compares a particular closing price of a security to a range of its prices over a certain period of time. Stochastic oscillators are based on the idea that closing prices should confirm the trend. For Stochastic (STOCH), there are four different lines: K^{Fast}, D^{Fast}, K^{Slow}, and D^{Slow}. D is the signal line usually drawn over its corresponding K function:

$$K^{\text{Fast}}(T_K) = \frac{P_t - P^L_{T_K}}{P^H_{T_K} - P^L_{T_K}} * 100$$

$$D^{\text{Fast}}(T_{\text{FastD}}) = \text{MA}(T_{\text{FastD}})[K^{\text{Fast}}]$$

$$K^{\text{Slow}}(T_{\text{SlowK}}) = \text{MA}(T_{\text{SlowK}})[K^{\text{Fast}}]$$

$$D^{\text{Slow}}(T_{\text{SlowD}}) = \text{MA}(T_{\text{SlowD}})[K^{\text{Slow}}]$$

$P^L_{T_K}$, $P^H_{T_K}$, and $P^L_{T_K}$ are the extreme values of the last T_K period. K^{Slow} and D^{Fast} are equivalent when using the same period. We obtain the series shown in *Figure A.11*, as follows:

```
slowk, slowd = talib.STOCH(df.high,
                           df.low,
                           df.close,
                           fastk_period=14,
                           slowk_period=3,
                           slowk_matype=0,
                           slowd_period=3,
                           slowd_matype=0)
df['STOCH'] = slowd / slowk
```

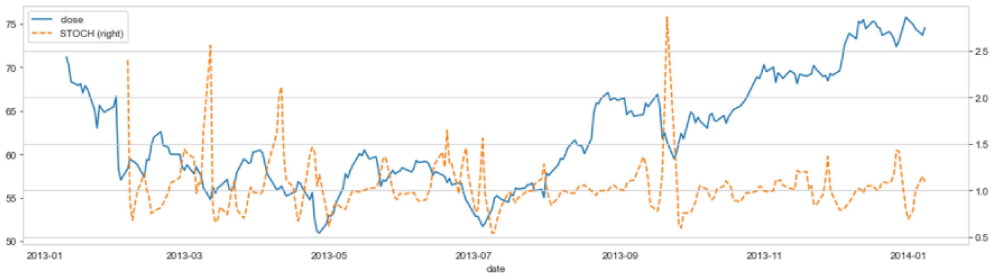

Figure A.11: STOCH series for the AAPL stock price

Ultimate oscillator

The **ultimate oscillator** (**ULTOSC**) measures the average difference between the current close and the previous lowest price over three timeframes—with the default values 7, 14, and 28—to avoid overreacting to short-term price changes and incorporate short-, medium-, and long-term market trends.

It first computes the buying pressure, BP_t, then sums it over the three periods T_1, T_2, and T_3, normalized by the true range (TR_t):

$$\mathrm{BP}_t = P_t^{\mathrm{Close}} - \min\left(P_{t-1}^{\mathrm{Close}}, P_t^{\mathrm{Low}}\right)$$

$$\mathrm{TR}_t = \max\left(P_{t-1}^{\mathrm{Close}}, P_t^{\mathrm{High}}\right) - \min\left(P_{t-1}^{\mathrm{Close}}, P_t^{\mathrm{Low}}\right)$$

ULTOSC is then computed as a weighted average over the three periods, as follows:

$$\mathrm{Avg}_t(T) = \frac{\sum_{i=0}^{T-1} \mathrm{BP}_{t-i}}{\sum_{i=0}^{T-1} \mathrm{TR}_{t-i}}$$

$$\mathrm{ULTOSC}_t = 100 * \frac{4\mathrm{Avg}_t(7) + 2\mathrm{Avg}_t(14) + \mathrm{Avg}_t(28)}{4 + 2 + 1}$$

The following plot shows the result of this:

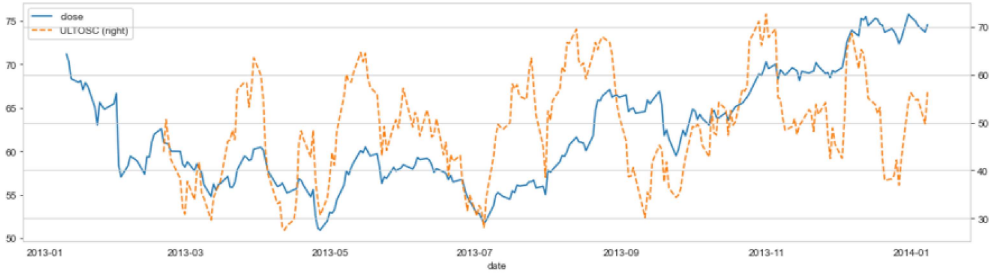

Figure A.12: ULTOSC series for the AAPL stock price

Williams %R

Williams %R, also known as the **Williams Percent Range**, is a momentum indicator that moves between 0 and -100 and measures overbought and oversold levels to identify entry and exit points. It is similar to the stochastic oscillator and compares the current closing price P_t^{Close} to the range of highest (P_T^{High}) and lowest (P_T^{Low}) prices over the last T periods (typically 14). The indicators are computed as follows, and the result is shown in the following chart:

$$\text{WILLR}_t = \frac{P_T^{\text{High}} - P_t^{\text{Close}}}{P_T^{\text{High}} - P_T^{\text{Low}}}$$

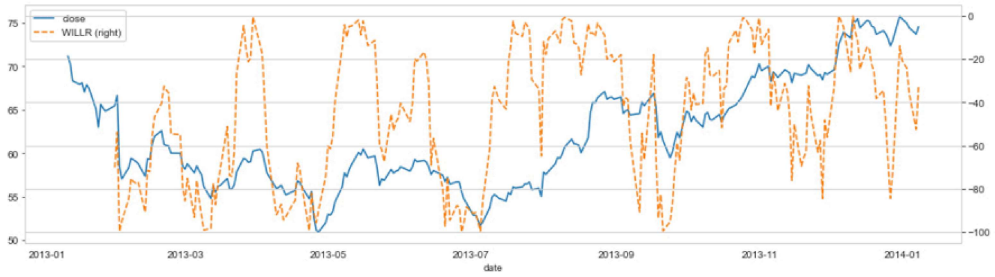

Figure A.13: WILLR series for the AAPL stock price

Volume and liquidity indicators

Risk factors that focus on volume and liquidity incorporate metrics like turnover, dollar volume, or market capitalization. TA-Lib implements three indicators, the first two of which are closely related:

Function	Name
AD	Chaikin A/D Line
ADOSC	Chaikin A/D Oscillator

OBV	On Balance Volume

Also see *Chapter 20, Autoencoders for Conditional Risk Factors and Asset Pricing*, where we use the Amihud Illiquidity indicator to measure a rolling average ratio between absolute returns and the dollar volume.

Chaikin accumulation/distribution line and oscillator

The **Chaikin advance/decline** (**AD**) or **accumulation/distribution** (**AD**) line is a volume-based indicator designed to measure the cumulative flow of money into and out of an asset. The indicator assumes that the degree of buying or selling pressure can be determined by the location of the close, relative to the high and low for the period. There is buying (selling) pressure when a stock closes in the upper (lower) half of a period's range. The intention is to signal a change in direction when the indicator diverges from the security price.

The A/D line is a running total of each period's **money flow volume** (**MFV**). It is calculated as follows:

1. Compute the **money flow index** (**MFI**) as the relationship of the close to the high-low range
2. Multiply the MFI by the period's volume V_t to come up with the MFV
3. Obtain the AD line as the running total of the MFV:

$$\text{MFI}_t = \frac{P_t^{\text{Close}} - P_t^{\text{Low}}}{P_t^{\text{High}} - P_t^{\text{Low}}}$$

$$\text{MFV}_t = \text{MFI}_t \times V_t$$

$$\text{AD}_t = \text{AD}_{t-1} + \text{MFV}_t$$

The **Chaikin A/D oscillator** (**ADOSC**) is the MACD indicator that's applied to the Chaikin AD line. The Chaikin oscillator intends to predict changes in the AD line.

It is computed as the difference between the 3-day EMA and the 10-day EMA of the AD line. The following chart shows the ADOSC series:

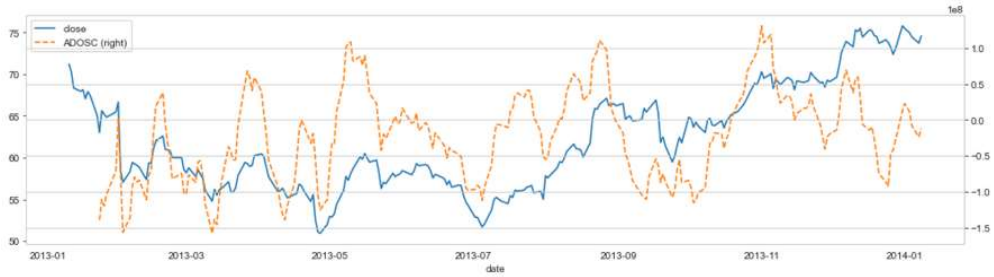

Figure A.14: ADOSC series for the AAPL stock price

On-balance volume

The **on-balance volume** (**OBV**) indicator is a cumulative momentum indicator that relates volume to price change. It assumes that OBV changes precede price changes because smart money can be seen flowing into the security by a rising OBV. When the public then follows, both the security and OBV will rise.

The current OBV_t is computed by adding (subtracting) the current volume to (from) the last $\text{OBV}_{t\text{-}1}$ if the security closes higher (lower) than the previous close:

$$\text{OBV}_t = \begin{cases} \text{OBV}_{t-1} + V_t & \text{if } P_t > P_t \\ \text{OBV}_{t-1} - V_t & \text{if } P_t < P_{t-1} \\ \text{OBV}_{t-1} & \text{otherwise} \end{cases}$$

Volatility indicators

Volatility indicators include stock-specific measures like the rolling (normalized) standard deviation of asset prices and returns. It also includes broader market measures like the Chicago Board Options Exchange's

CBOE **volatility index** (**VIX**), which is based on the implied volatility of S&P 500 options.

TA-Lib implements both normalized and averaged versions of the true range indicator.

Average true range

The **average true range** (**ATR**) indicator shows the volatility of the market. It was introduced by Wilder (1978) and has been used as a component of numerous other indicators since. It aims to anticipate changes in trend such that the higher its value, the higher the probability of a trend change; the lower the indicator's value, the weaker the current trend.

ATR is computed as the simple moving average for a period T of the **true range** (**TRANGE**), which measures volatility as the absolute value of the largest recent trading range:

$$\text{TRANGE}_t = \max\left[P_t^{\text{High}} - P_t^{\text{low}}, \left|P_t^{\text{High}} - P_{t-1}^{\text{Close}}\right|, \left|P_t^{\text{low}} - P_{t-1}^{\text{Close}}\right|\right]$$

The resulting series is shown in the following plot:

Figure A.15: ATR series for the AAPL stock price

Normalized average true range

TA-Lib also offers a normalized ATR that permits comparisons across assets. The **normalized average true range** (**NATR**) is computed as follows:

$$\text{NATR}_t = \frac{\text{ATR}_t(T)}{P_t^{\text{Close}}} * 100$$

Normalization makes the ATR more relevant for long-term analysis where the price changes substantially and for cross-market or cross-security comparisons.

Fundamental risk factors

Commonly used measures of risk include the exposure of asset returns to the returns of portfolios designed to represent fundamental factors. We introduced the five-factor model by Fama and French (2015) and showed how to estimate factor loadings and risk factor premia using two-state Fama-Macbeth regressions in *Chapter 7*, *Linear Models – From Risk Factors to Return Forecasts*.

To estimate the relationship between the price of security and the forces included in the five-factor model such as firm size, value-versus-growth dynamic, investment policy and profitability, in addition to the broad market, we can use the portfolio returns provided by Kenneth French's data library as exogenous variables in a rolling linear regression.

The following example accesses the data using the `pandas_datareader` module (see *Chapter 2*, *Market and Fundamental Data – Sources and Techniques*). It then computes the regression coefficients for windows of 21, 63, and 252 trading days:

```
factor_data = (web.DataReader('F-F_Research_Data_5_Factors_2x3_daily', 'famafrencl
                              start=2005)[0].rename(columns={'Mkt-RF': 'MARKET'})
factor_data.index.names = ['date']
factors = factor_data.columns[:-1]
t = 1
ret = f'ret_{t:02}'
windows = [21, 63, 252]
for window in windows:
    print(window)
    betas = []
    for ticker, df in data.groupby('ticker', group_keys=False):
```

```
            model_data = df[[ret]].merge(factor_data, on='date').dropna()
            model_data[ret] -= model_data.RF
            rolling_ols = RollingOLS(endog=model_data[ret],
                                     exog=sm.add_constant(model_data[factors]),
                                     window=window)
            factor_model = rolling_ols.fit(params_only=True).params.rename(
                columns={'const':'ALPHA'})
            result = factor_model.assign(ticker=ticker).set_index(
                'ticker', append=True).swaplevel()
            betas.append(result)
        betas = pd.concat(betas).rename(columns=lambda x: f'{x}_{window:02}')
        data = data.join(betas)
```

The risk factors just described are commonly used and also known as **smart beta factors** (see *Chapter 1*, *Machine Learning for Trading – From Idea to Execution*). In addition, hedge funds have started to resort to alpha factors derived from large-scale data mining exercises, which we'll turn to now.

WorldQuant's quest for formulaic alphas

We introduced WorldQuant in *Chapter 1*, *Machine Learning for Trading – From Idea to Execution*, as part of a trend toward crowd-sourcing investment strategies. WorldQuant maintains a virtual research center where quants worldwide compete to identify **alphas**. These alphas are trading signals in the form of computational expressions that help predict price movements, just like the common factors described in the previous section.

These **formulaic alphas** translate the mechanism to extract the signal from data into code, and they can be developed and tested individually with the goal to integrate their information into a broader automated strategy (Tulchinsky 2019). As stated repeatedly throughout this book, mining for signals in large datasets is prone to multiple testing bias and false discoveries. Regardless of these important caveats, this approach represents a modern alternative to the more conventional features presented in the previous section.

Kakushadze (2016) presents 101 examples of such alphas, 80 percent of which were used in a real-world trading system at the time. It defines a range of functions that operate on cross-sectional or time-series data and can be combined, for example, in nested form.

The notebook `101_formulaic_alphas` shows how to implement these functions using pandas and NumPy, and also illustrates how to compute around 80 of these formulaic alphas for which we have input data (we lack, for example, accurate historical sector information).

Cross-sectional and time-series functions

The building blocks of the formulaic alphas proposed by Kakushadze (2016) are relatively simple expressions that compute over longitudinal or cross-sectional data that are readily implemented using pandas and NumPy.

The cross-sectional functions include ranking and scaling, as well as the group-wise normalization of returns, where the groups are intended to represent sector information at different levels of granularity:

We can directly translate the ranking function into a pandas expression, using a DataFrame as an argument in the format *number of period* × *number of tickers*, as follows:

```
def rank(df):
    """Return the cross-sectional percentile rank
     Args:
         :param df: tickers in columns, sorted dates in rows.
     Returns:
         pd.DataFrame: the ranked values
     """
    return df.rank(axis=1, pct=True)
```

There are also several time-series functions that will likely be familiar:

Function	Definition
`ts_{O}(x, d)`	Operator *O* applied to the time series for the past *d* days; non-integer number of

	days *d* converted to floor(*d*).
`ts_lag(x, d)`	Value of *x*, *d* days ago.
`ts_delta(x, d)`	Difference between the value of *x* today and *d* days ago.
`ts_rank(x, d)`	Rank over the past *d* days.
`ts_mean(x, d)`	Simple moving average over the past *d* days.
`ts_weighted_mean(x, d)`	Weighted moving average over the past *d* days with linearly decaying weights *d*, *d* – 1, …, 1 (rescaled to sum up to 1).
`ts_sum(x, d)`	Rolling sum over the past *d* days.
`ts_product(x, d)`	Rolling product over the past *d* days.
`ts_stddev(x, d)`	Moving standard deviation over the past *d* days.
`ts_max(x, d)`, `ts_min(x, d)`	Rolling maximum/minimum over the past *d* days.
`ts_argmax(x, d)`, `ts_argmin(x, d)`	Day of ts_*max(x, d), ts_min*(*x*, *c*).
`ts_correlation(x, y, d)`	Correlation of *x* and *y* for the past *d* days.

These time-series functions are also straightforward to implement using pandas' rolling window functionality. For the rolling weighted mean, for example, we can combine pandas with TA-Lib, as demonstrated in the previous section:

```
def ts_weighted_mean(df, period=10):
    """
```

```
    Linear weighted moving average implementation.
    :param df: a pandas DataFrame.
    :param period: the LWMA period
    :return: a pandas DataFrame with the LWMA.
    """
    return (df.apply(lambda x: WMA(x, timeperiod=period)))
```

To create the rolling correlation function, we provide two DataFrames containing time series for different tickers in the columns:

```
def ts_corr(x, y, window=10):
    """
    Wrapper function to estimate rolling correlations.
    :param x, y: pandas DataFrames.
    :param window: the rolling window.
    :return: DataFrame with time-series min for past 'window' days.
    """
    return x.rolling(window).corr(y)
```

In addition, the expressions use common operators, as we will see as we turn to the formulaic alphas that each combine several of the preceding functions.

Formulaic alpha expressions

To illustrate the computation of the alpha expressions, we need to create the following input tables using the sample of the 500 most-traded stocks from 2007-2016 from the previous section (see the notebook `sample_selection` for details on data preparation). Each table contains columns of time series for individual tickers:

Variable	Description
`returns`	Daily close-to-close returns
`open`, `close`, `high`, `low`, `volume`	Standard definitions for daily price and volume data
`vwap`	Daily volume-weighted average price

adv(d)	Average daily dollar volume for the past *d* days

Our data does not include the daily volume-weighted average price required by many alpha expressions. To be able to demonstrate their computation, we very roughly approximate this value using the simple average of the daily open, high, low, and close prices.

Contrary to the common alphas presented in the previous section, the formulaic alphas do not come with an economic interpretation of the risk exposure they represent. We will now demonstrate a few simply numbered instances.

Alpha 001

The first alpha expression is formulated as follows:

```
rank(ts_argmax(power(((returns < 0) ? ts_std(returns, 20) : close), 2.), 5))
```

The ternary operator `a ? b : c` executes *b* if *a* evaluates to `true`, and *c* otherwise. Thus, if the daily returns are positive, it squares the 20-day rolling standard deviation; otherwise, it squares the current close price. It then proceeds to rank the assets by the index of the day that shows the maximum for this value.

Using *c* and *r* to represent the close and return inputs, the alpha translates into Python using the previous functions and pandas methods, like so:

```
def alpha001(c, r):
    """(rank(ts_argmax(power(((returns < 0)
        ? ts_std(returns, 20)
        : close), 2.), 5)) -0.5)"""
    c[r < 0] = ts_std(r, 20)
    return (rank(ts_argmax(power(c, 2), 5)).mul(-.5)
            .stack().swaplevel())
```

For the 10-year sample of 500 stocks, the distribution of Alpha 001 values and its relationship with one-day forward returns looks as follows:

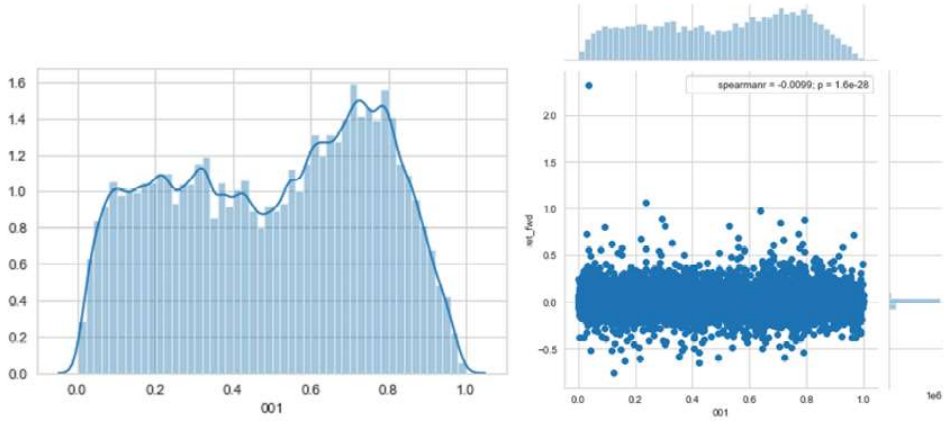

Figure A.16: Alpha 001 histogram and scatter plot

The **information coefficient** (**IC**) is fairly low, yet it is statistically significant at -0.0099 and the **mutual information** (**MI**) estimate yields 0.0129 (see *Chapter 4*, *Financial Feature Engineering – How to Research Alpha Factors*, and the notebook `101_formulaic_alphas`, for implementation details).

Alpha 054

Our second expression is the ratio of the difference between the low and the close price and the low and the high price, each multiplied by the open and close, respectively, raised to the fifth power:

```
-(low - close) * power(open, 5) / ((low - high) * power(close, 5))
```

Similarly, the translation into pandas is straightforward. We use `o`, `h`, `l`, and c to represent the DataFrames containing the respective price series for each ticker in the 500 columns:

```
def alpha054(o, h, l, c):
    """-(low - close) * power(open, 5) / ((low - high) * power(close, 5))"""
    return (l.sub(c).mul(o.pow(5)).mul(-1)
            .div(l.sub(h).replace(0, -0.0001).mul(c ** 5))
            .stack('ticker')
            .swaplevel())
```

In this case, the IC is significant at 0.025, while the MI score is lower at 0.005.

We will now take a look at how these different types of alpha factors compare from a univariate and a multivariate perspective.

Bivariate and multivariate factor evaluation

To evaluate the numerous factors, we rely on the various performance measures introduced in this book, including the following:

- Bivariate measures of the signal content of a factor with respect to the one-day forward returns
- Multivariate measures of feature importance for a gradient boosting model trained to predict the one-day forward returns using all factors
- Financial performance of portfolios invested according to factor quantiles using Alphalens

We will first discuss the bivariate metrics and then turn to the multivariate metrics; we will conclude by comparing the results. See the notebook `factor_evaluation` for the relevant code examples and additional exploratory analysis, such as the correlation among the factors, which we'll omit here.

Information coefficient and mutual information

We will use the following bivariate metrics, which we introduced in *Chapter 4*, *Financial Feature Engineering – How to Research Alpha Factors*:

- The IC measured as the Spearman rank correlation
- The MI score computed using `mutual_info_regression`, provided by scikit-learn

The MI score uses a sample of 100,000 observations to limit the computational cost of the nearest neighbor computations. Both metrics are otherwise easy to compute and have been used repeatedly; see the notebook for implementation details. We will see, however, that they can yield quite different results.

Feature importance and SHAP values

To measure the predictive relevance of a feature given all other available factors, we can train a LightGBM gradient boosting model with default settings to predict the forward returns using all of the (approximately) 130 factors. The model uses 8.5 years of data to train 104 trees using early stopping. We will obtain test predictions for the last year of data, which yield a global IC of 3.40 and a daily average of 2.01.

We will then proceed to compute the feature importance and **SHapley Additive exPlanation** (**SHAP**) values, as described in *Chapter 12*, *Boosting Your Trading Strategy*; see the notebook for details. The influence plot in *Figure A.17* highlights how the values of the 20 most important features impact the model's predictions positively or negatively relative to the model's default output. In SHAP value terms, alphas 054 and 001 are among the top five factors:

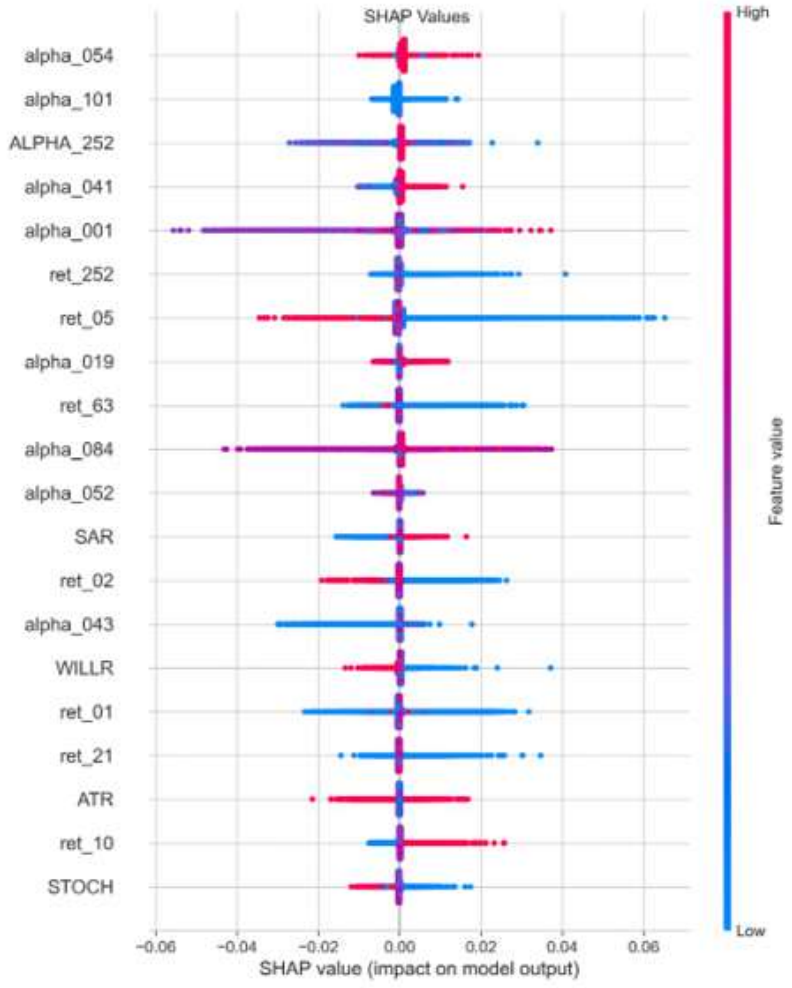

Figure A.17: SHAP values for common and formulaic alphas

Now, let's compare how the different metrics rate our factors.

Comparison – the top 25 features for each metric

The rank correlation among SHAP values and conventional feature importance measured as the weighted contribution of a feature to the reduction of the model's loss function is high at 0.89. It is also substantial between SHAP values and both univariate metrics at around 0.5.

Interestingly, though, MI and IC disagree significantly in their feature rankings with a correlation of only 0.16, as shown in the following diagram:

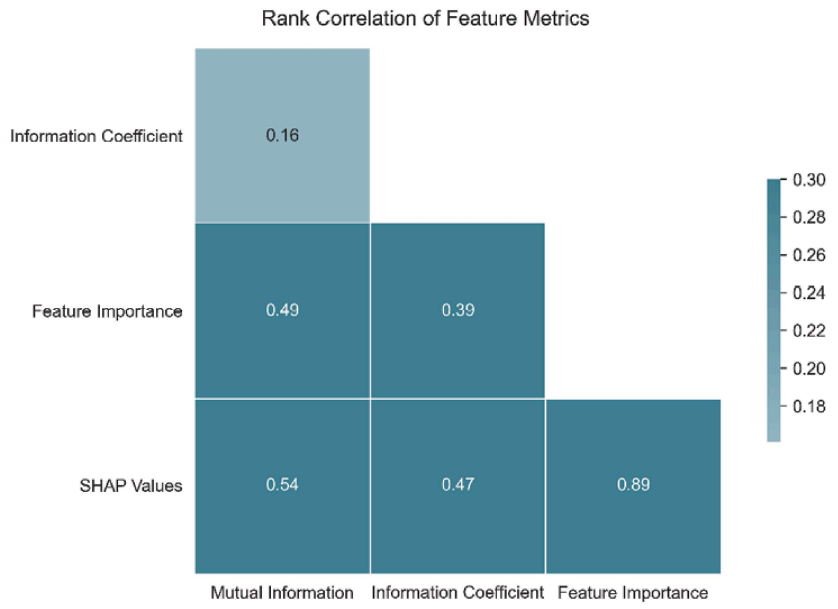

Figure A.18: Rank correlation of performance metrics

Figure A.19 displays the top 25 features according to each metric. Except for the MI score, which prefers the "common" alpha factors, features from both sources are ranked highly:

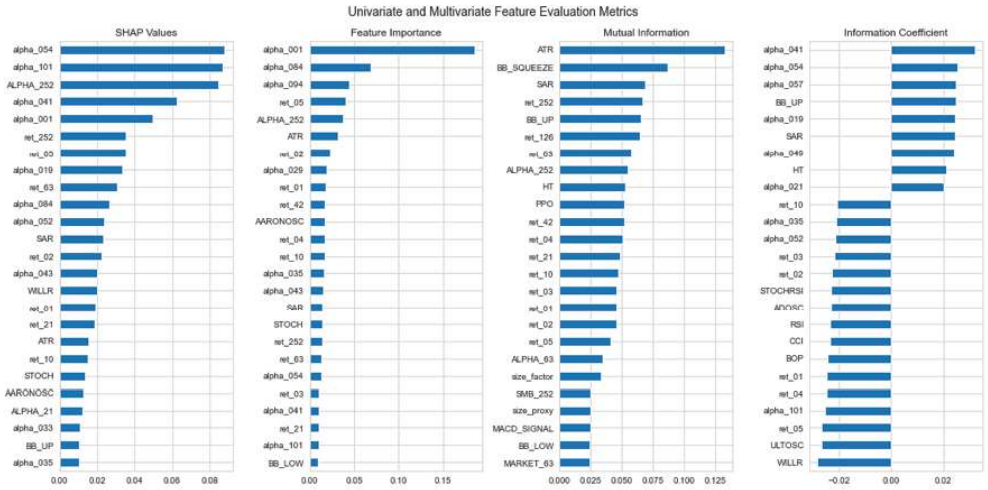

Figure A.19: Top 25 features for each performance metric

It is not immediately apparent why MI disagrees with the other metrics and why few of the features it assigns a high score play a significant role in the gradient boosting model. A possible explanation is that the computation uses only a 10 percent sample and the scores appear sensitive to the sample size.

Financial performance – Alphalens

Finally, we mostly care about the value of the trading signals emitted by an alpha factor. As introduced in *Chapter 4*, *Financial Feature Engineering – How to Research Alpha Factors*, and demonstrated repeatedly, Alphalens evaluates factor performance on a standalone basis.

The notebook `alphalens_analysis` lets you select an individual factor and compute how portfolios invested for a given horizon according to how factor quantile values would have performed.

The example in *Figure A.20* shows the result for Alpha 54; while portfolios in the top and bottom quintiles did achieve a 1.5 bps average spread on a daily basis, the cumulative returns of a long-short portfolio were negative:

Figure A.20: Alphalens performance metric for Alpha 54

Feel free to use the notebook as a template to evaluate the sample factors or others of your own choosing more systematically.