2- Feature Calculation

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
%matplotlib inline

from pathlib import Path
import numpy as np
import pandas as pd
import pandas_datareader.data as web

import statsmodels.api as sm
from statsmodels.regression.rolling import RollingOLS
from sklearn.preprocessing import scale
import talib

import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
sns.set_style('whitegrid')
idx = pd.IndexSlice
deciles = np.arange(.1, 1, .1).round(1)
```

Load Data

In [4]:
```
DATA_STORE = Path('data', 'stock_prices.h5')
```

In [5]:
```
DATA_STORE
```

Out[5]:
```
WindowsPath('data/stock_prices.h5')
```

In [6]:
```
with pd.HDFStore(DATA_STORE) as store:
```

```
    data = (store['us_stocks']
            .loc[idx[:, '2006':'2016'], :]  # slice 11 years from 2nd level of MultiIndex using pd.IndexSlice
            .unstack('ticker') # move first index level 'ticker' into the columns
            .sort_index()
            .fillna(method='ffill', limit=5) # fill up to five days of missing data with latest
            .stack('ticker') # move Index level 'ticker' back into the rows
            .swaplevel()     # swap levels of row index so we're back at (ticker, data)
            .dropna()        # remove missing values
            .sort_index())
```

In [7]:
```
data.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 7739201 entries, ('A', Timestamp('2006-01-03 00:00:00')) to ('ZUMZ', Timestamp('2016-12-30 00:00:00'))
Data columns (total 12 columns):
 #   Column       Non-Null Count    Dtype  
---  ------       --------------    -----  
 0   open         7739201 non-null  float64
 1   high         7739201 non-null  float64
 2   low          7739201 non-null  float64
 3   close        7739201 non-null  float64
 4   volume       7739201 non-null  float64
 5   ex-dividend  7739201 non-null  float64
 6   split_ratio  7739201 non-null  float64
 7   adj_open     7739201 non-null  float64
 8   adj_high     7739201 non-null  float64
 9   adj_low      7739201 non-null  float64
 10  adj_close    7739201 non-null  float64
 11  adj_volume   7739201 non-null  float64
dtypes: float64(12)
memory usage: 738.3+ MB
```

Remove outliers based on daily returns

In [9]:
```
daily_returns = data.groupby('ticker').close.pct_change() # returns computed based on closing prices
```

In [10]:
```
daily_returns.describe(percentiles=[.00001, .0001, .001, .999, .9999, .99999]).iloc[1:]
```

Out[10]:
```
mean         0.001053
std          0.221775
min         -0.990476
0.001%      -0.747672
0.01%       -0.486981
0.1%        -0.196699
50%          0.000000
99.9%        0.233333
99.99%       0.679408
99.999%      9.794364
max        393.444444
Name: close, dtype: float64
```

In [11]:
```
outliers = daily_returns[(daily_returns < daily_returns.quantile(.00001)) |
                         (daily_returns > daily_returns.quantile(.99999))]
```

In [13]:
```
data = data.drop(outliers.index.unique('ticker'), level='ticker')
```

Select 500 most-traded stocks prior to 2017

Compute the dollar volume as the product of the adjusted close price and the adjusted volume:

In [14]:
```
dv = data.close.mul(data.volume)
```

Compute the daily dollar volume rank, average the ranks by ticker, and select the 500 tickers with the highest average rank (equivalent to the lowest rank value).

In [15]:
```
top500 = (dv.groupby(level='date')
          .rank(ascending=False)
          .unstack('ticker')
          .dropna(thresh=8*252, axis=1)
          .mean()
          .nsmallest(500))
```

In [17]:
```
to_drop = data.index.unique('ticker').difference(top500.index)
```

In [18]:
```
len(to_drop)
```

Out[18]:
```
2562
```

In [19]:
```
data = data.drop(to_drop, level='ticker')
```

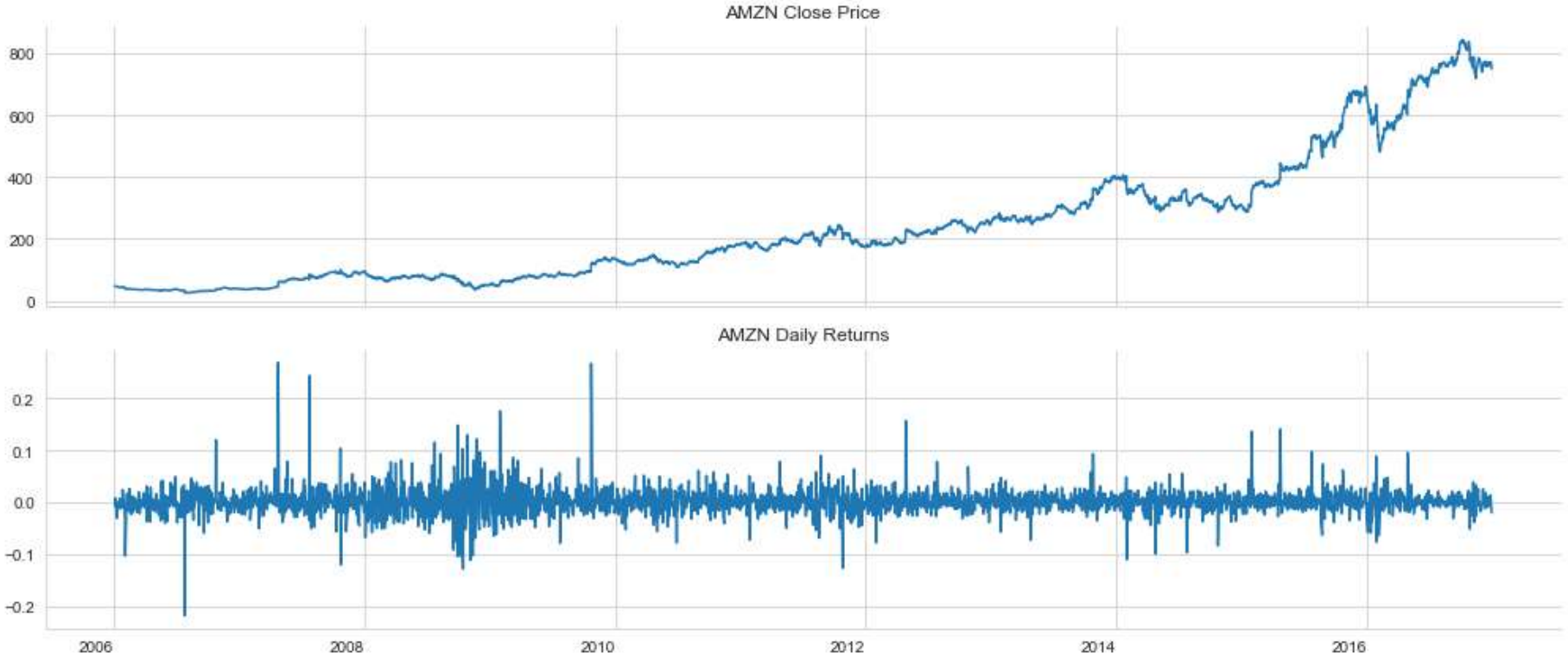
In [20]:
```
data.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1404084 entries, ('A', Timestamp('2006-01-03 00:00:00')) to ('ZMH', Timestamp('2015-07-06 00:00:00'))
Data columns (total 12 columns):
 #   Column       Non-Null Count    Dtype  
---  ------       --------------    -----  
 0   open         1404084 non-null  float64
 1   high         1404084 non-null  float64
 2   low          1404084 non-null  float64
 3   close        1404084 non-null  float64
 4   volume       1404084 non-null  float64
 5   ex-dividend  1404084 non-null  float64
 6   split_ratio  1404084 non-null  float64
 7   adj_open     1404084 non-null  float64
 8   adj_high     1404084 non-null  float64
 9   adj_low      1404084 non-null  float64
 10  adj_close    1404084 non-null  float64
 11  adj_volume   1404084 non-null  float64
dtypes: float64(12)
memory usage: 134.1+ MB
```

Sample price data to illustrate factors

In [23]:
```
ticker = 'AMZN'

fig, axes = plt.subplots(nrows=2, figsize=(14, 6), sharex=True) # create figure with 2 axes
s = data.loc[ticker, 'close'] # select closing prices for ticker
s.plot(rot=0, ax=axes[0], title=f'{ticker} Close Price') # plot prices on first axes
s.pct_change().plot(rot=0, ax=axes[1], title=f'{ticker} Daily Returns') # add returns to second axis
axes[1].set_xlabel('')
```

```
sns.despine()
fig.tight_layout()
```



In [24]:
```
ticker = 'AMZN'
price_sample = data.loc[idx[ticker, :], :].reset_index('ticker', drop=True)
```

In [25]:
```
price_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2845 entries, 2006-01-03 to 2016-12-30
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   open          2845 non-null   float64
 1   high          2845 non-null   float64
 2   low           2845 non-null   float64
 3   close         2845 non-null   float64
 4   volume        2845 non-null   float64
 5   ex-dividend   2845 non-null   float64
 6   split_ratio   2845 non-null   float64
 7   adj_open      2845 non-null   float64
 8   adj_high      2845 non-null   float64
 9   adj_low       2845 non-null   float64
 10  adj_close     2845 non-null   float64
 11  adj_volume    2845 non-null   float64
dtypes: float64(12)
memory usage: 288.9 KB
```

Group data by ticker

Feature Calculation

In [26]:
```
by_ticker = data.groupby(level='ticker')
```

Historical returns

In [27]:
```
T = [1, 3, 5, 10, 21, 42, 63, 126, 252]
```

In [28]:
```
for t in T:
    data[f'ret_{t:02}'] = by_ticker.close.pct_change(t) # compute returns for each ticker and period
```

Forward returns

In [29]:
```
data['ret_fwd'] = by_ticker.ret_01.shift(-1) # shift returns back in time (tomorrow's returns are today's fwd retur
data = data.dropna(subset=['ret_fwd'])
```

In [32]:
```
data.info(show_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1403584 entries, ('A', Timestamp('2006-01-03 00:00:00')) to ('ZMH', Timestamp('2015-07-02 00:00:00'))
Data columns (total 22 columns):
 #   Column       Non-Null Count    Dtype  
---  ------       --------------    -----  
 0   open         1403584 non-null  float64
 1   high         1403584 non-null  float64
 2   low          1403584 non-null  float64
 3   close        1403584 non-null  float64
 4   volume       1403584 non-null  float64
 5   ex-dividend  1403584 non-null  float64
 6   split_ratio  1403584 non-null  float64
 7   adj_open     1403584 non-null  float64
 8   adj_high     1403584 non-null  float64
 9   adj_low      1403584 non-null  float64
 10  adj_close    1403584 non-null  float64
 11  adj_volume   1403584 non-null  float64
 12  ret_01       1403084 non-null  float64
 13  ret_03       1402084 non-null  float64
 14  ret_05       1401084 non-null  float64
 15  ret_10       1398584 non-null  float64
 16  ret_21       1393084 non-null  float64
 17  ret_42       1382584 non-null  float64
 18  ret_63       1372084 non-null  float64
 19  ret_126      1340584 non-null  float64
 20  ret_252      1277584 non-null  float64
 21  ret_fwd      1403584 non-null  float64
dtypes: float64(22)
memory usage: 241.2+ MB
```

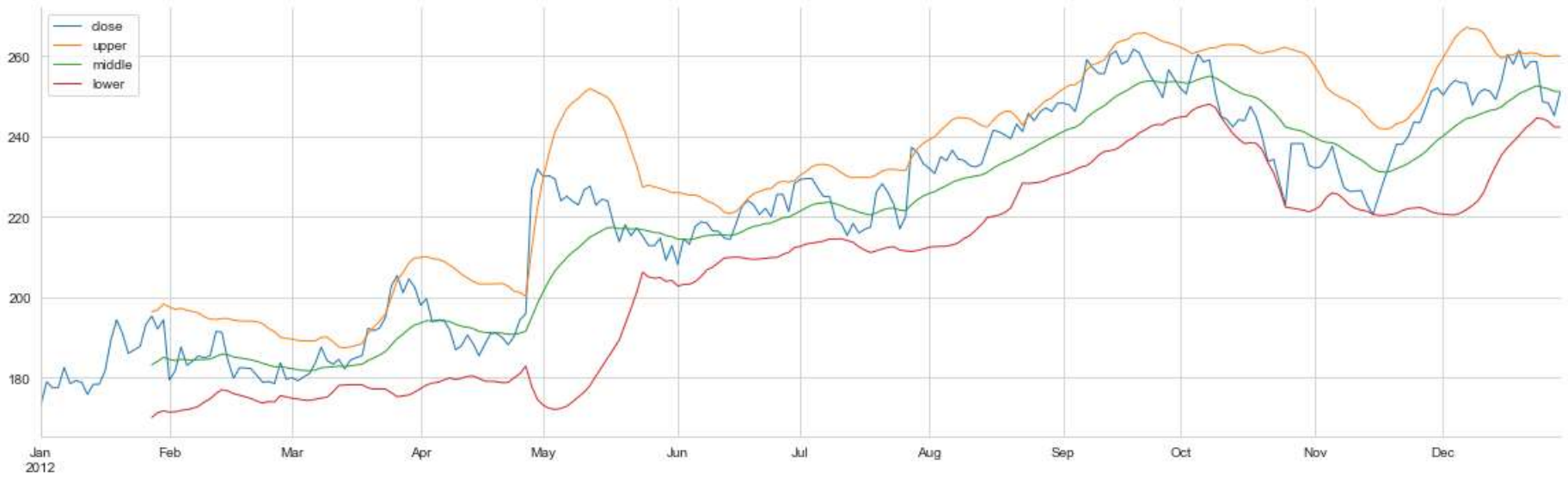
Bollinger Bands

In [33]:
```
df = price_sample.loc['2012', ['close']]
```

In [34]:
```
s = talib.BBANDS(df.close,   # Number 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
                 )
```

In [35]:
```
bb_bands = ['upper', 'middle', 'lower']
```

In [36]:
```
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)

ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```



Normalized squeeze & mean reversion indicators

In [37]:
```
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', rot=0)

plt.legend()
ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```

In [38]:
```
def compute_bb_indicators(close, timeperiod=20, matype=0):
    high, mid, low = talib.BBANDS(close,
                                  timeperiod=20,
                                  matype=matype)
    bb_up = high / close -1 # normalize with respect to close
    bb_low = low / close -1 # normalize with respect to close
    squeeze = (high - low) / close
    return pd.DataFrame({'BB_UP': bb_up,
                         'BB_LOW': bb_low,
                         'BB_SQUEEZE': squeeze},
                        index=close.index)
```

In [39]:
```
data = (data.join(data
                  .groupby(level='ticker')
```

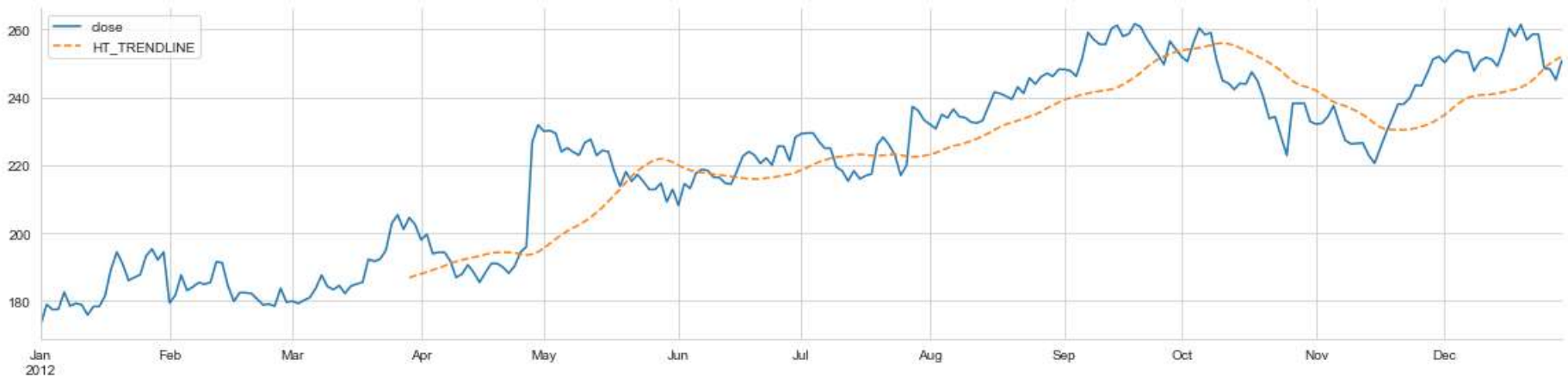
```
                    .close
                    .apply(compute_bb_indicators)))
```

Visualize Distribution

In [40]:
```
bb_indicators = ['BB_UP', 'BB_LOW', 'BB_SQUEEZE']
```

In [43]:
```
df = price_sample.loc['2012', ['close']]
df['HT_TRENDLINE'] = talib.HT_TRENDLINE(df.close)
```

In [44]:
```
ax = df.plot(figsize=(16, 4), style=['-', '--'], rot=0)

ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```



In [45]:
```
data['HT'] = (data
              .groupby(level='ticker', group_keys=False)
              .close
              .apply(talib.HT_TRENDLINE)
              .div(data.close).sub(1))
```

In [47]:
```
df = price_sample.loc['2012', ['close', 'high', 'low']]
df['SAR'] = talib.SAR(df.high, df.low,
```

```
                    acceleration=0.02, # common value
                    maximum=0.2)
```

In [48]:
```
ax = df[['close', 'SAR']].plot(figsize=(16, 4), style=['-', '--'], title='Parabolic SAR')
ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```

Momentum Indicators

Function	Name
PLUS_DM	Plus Directional Movement
MINUS_DM	Minus Directional Movement
PLUS_DI	Plus Directional Indicator
MINUS_DI	Minus Directional Indicator
DX	Directional Movement Index
ADX	Average Directional Movement Index
ADXR	Average Directional Movement Index Rating

Function	Name
APO	Absolute Price Oscillator
PPO	Percentage Price Oscillator
AROON	Aroon
AROONOSC	Aroon Oscillator
BOP	Balance Of Power
CCI	Commodity Channel Index
CMO	Chande Momentum Oscillator
MACD	Moving Average Convergence/Divergence
MACDEXT	MACD with controllable MA type
MACDFIX	Moving Average Convergence/Divergence Fix 12/26
MFI	Money Flow Index
MOM	Momentum
RSI	Relative Strength Index
STOCH	Stochastic
STOCHF	Stochastic Fast
STOCHRSI	Stochastic Relative Strength Index
TRIX	1-day Rate-Of-Change (ROC) of a Triple Smooth EMA
ULTOSC	Ultimate Oscillator
WILLR	Williams' %R

In [52]:
```
df = price_sample.loc['2012': '2013', ['high', 'low', 'close']]
```

In [53]:
```
df['PLUS_DM'] = talib.PLUS_DM(df.high, df.low, timeperiod=10)
```

```
df['MINUS_DM'] = talib.MINUS_DM(df.high, df.low, timeperiod=10)
```

In [54]:
```
ax = df[['close', 'PLUS_DM', 'MINUS_DM']].plot(figsize=(14, 4),
                                               secondary_y=[
                                                   'PLUS_DM', 'MINUS_DM'],
                                               style=['-', '--', '_'],
                                              rot=0)
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

In [55]:
```
df = price_sample.loc['2012': '2013', ['high', 'low', 'close']]
```

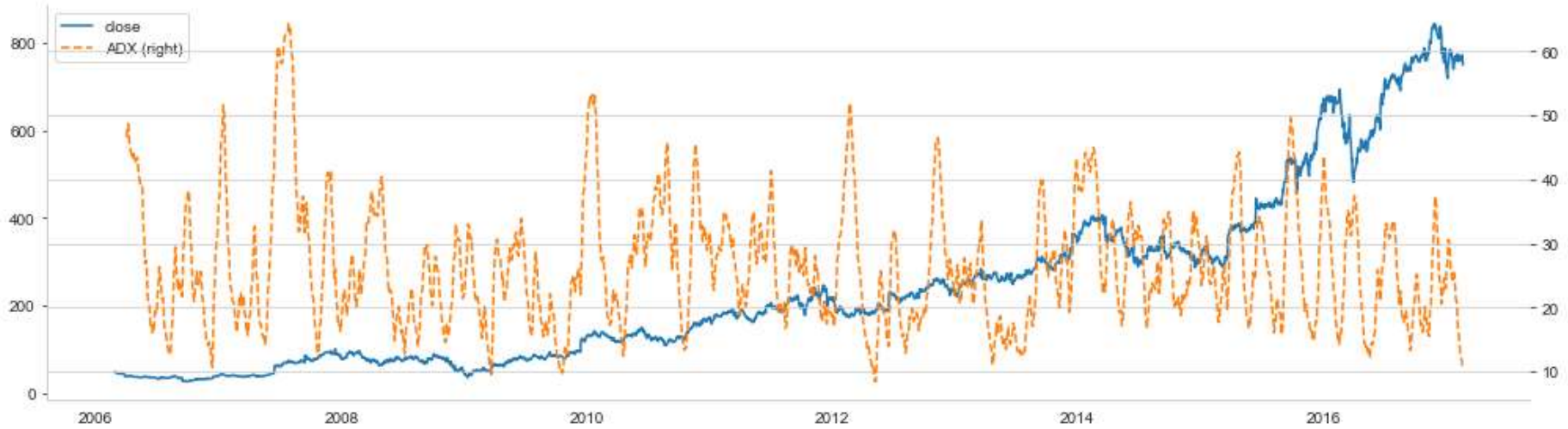
In [56]:
```
df['PLUS_DI'] = talib.PLUS_DI(df.high, df.low, df.close, timeperiod=14)
df['MINUS_DI'] = talib.MINUS_DI(df.high, df.low, df.close, timeperiod=14)
```

In [57]:
```
ax = df[['close', 'PLUS_DI', 'MINUS_DI']].plot(figsize=(14, 5), style=['-', '--', '_'], rot=0)

ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```

In [58]:
```
df = price_sample.loc[:, ['high', 'low', 'close']]
```

In [59]:
```
df['ADX'] = talib.ADX(df.high,
                      df.low,
                      df.close,
                      timeperiod=14)
```

In [60]:
```
ax = df[['close', 'ADX']].plot(figsize=(14, 4), secondary_y='ADX', style=['-', '--'], rot=0)
ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```



In [64]:
```
df = price_sample.loc[:, ['high', 'low', 'close']]
```

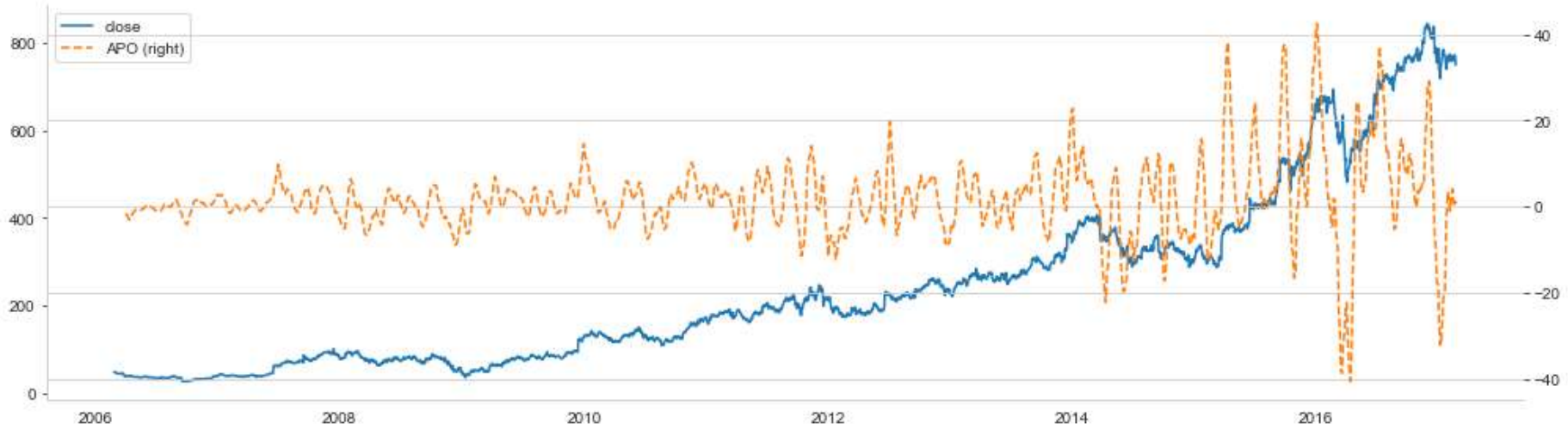
In [65]:
```
df['ADXR'] = talib.ADXR(df.high,
                        df.low,
                        df.close,
                        timeperiod=14)
```

In [66]:
```
ax = df[['close', 'ADXR']].plot(figsize=(14, 5),
                                secondary_y='ADX',
                                style=['-', '--'], rot=0)
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

In [70]:
```
df = price_sample.loc[:, ['close']]
```

In [71]:
```
df['APO'] = talib.APO(df.close,
                      fastperiod=12,
                      slowperiod=26,
                      matype=0)
```

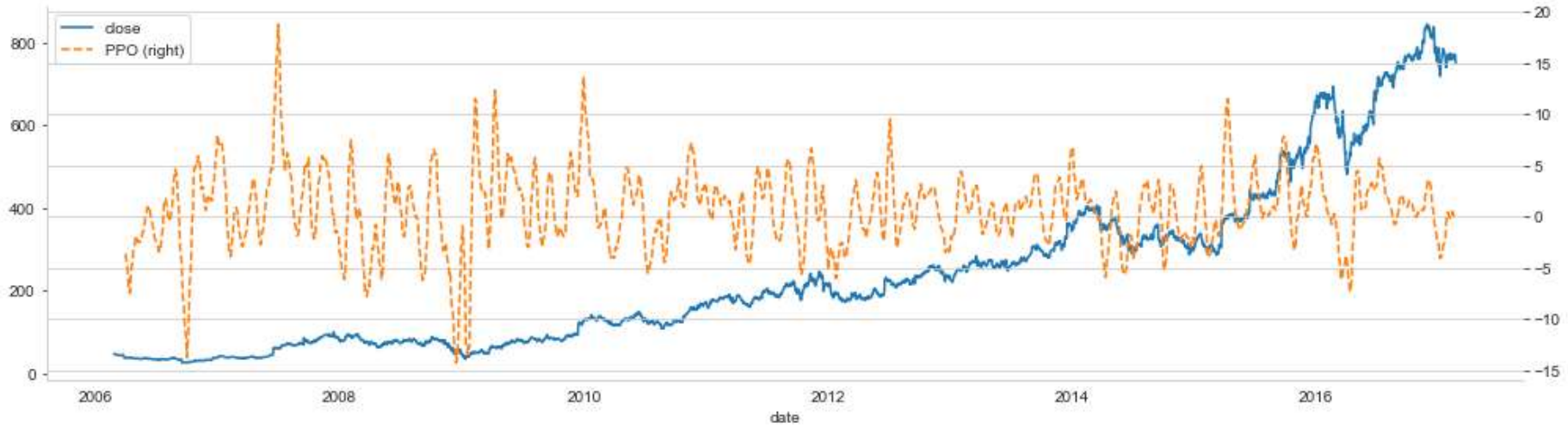
In [72]:
```
ax = df.plot(figsize=(14,4), secondary_y='APO', rot=0, style=['-', '--'])
ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```



In [73]:
```
df = price_sample.loc[:, ['close']]
```

In [74]:
```
df['PPO'] = talib.PPO(df.close,
                      fastperiod=12,
                      slowperiod=26,
                      matype=0)
```

In [75]:
```
ax = df.plot(figsize=(14,4), secondary_y=['APO', 'PPO'], rot=0,  style=['-', '--'])

ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```

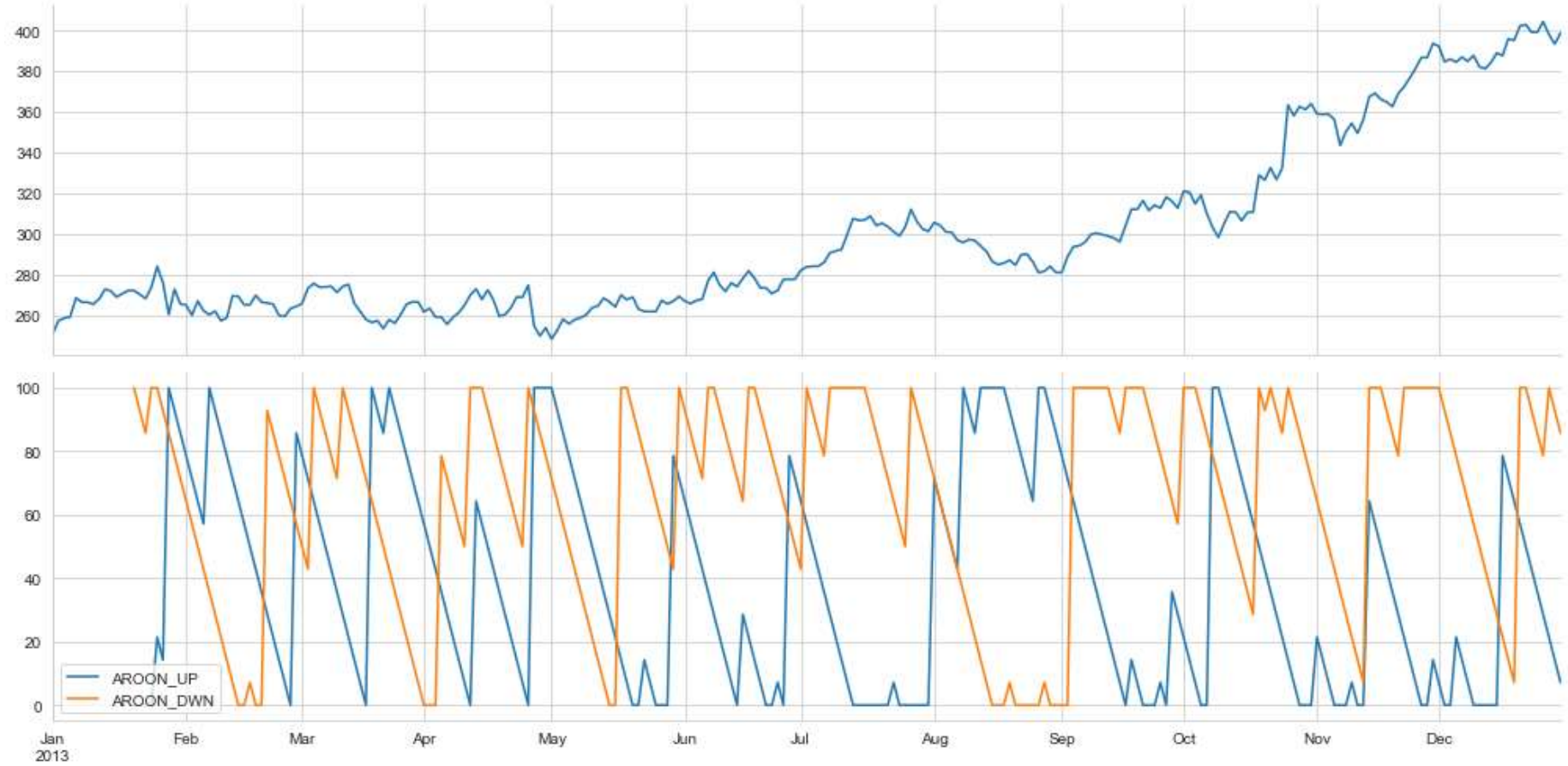


In [76]:
```
data['PPO'] = (data.groupby(level='ticker')
                .close
                .apply(talib.PPO,
                       fastperiod=12,
                       slowperiod=26,
                       matype=1))
```

In [78]:
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [79]:
```
aroonup, aroondwn = talib.AROON(high=df.high,
                                low=df.low,
                                timeperiod=14)
df['AROON_UP'] = aroonup
df['AROON_DWN'] = aroondwn
```

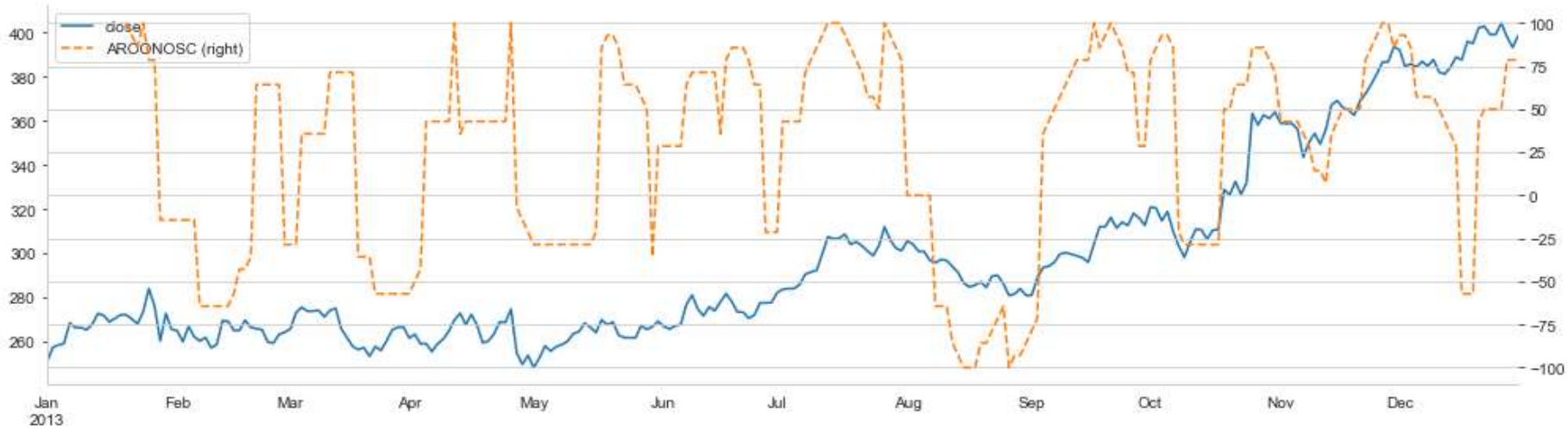
In [80]:
```
fig, axes = plt.subplots(nrows=2, figsize=(14, 7), sharex=True)
df.close.plot(ax=axes[0], rot=0)
df[['AROON_UP', 'AROON_DWN']].plot(ax=axes[1], rot=0)

axes[1].set_xlabel('')
sns.despine()
plt.tight_layout();
```



In [81]:
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [82]:
```
df['AROONOSC'] = talib.AROONOSC(high=df.high,
                                low=df.low,
                                timeperiod=14)
```

In [83]:
```
ax = df[['close', 'AROONOSC']].plot(figsize=(14,4), rot=0, style=['-', '--'], secondary_y='AROONOSC')
ax.set_xlabel('')
sns.despine()
plt.tight_layout();
```

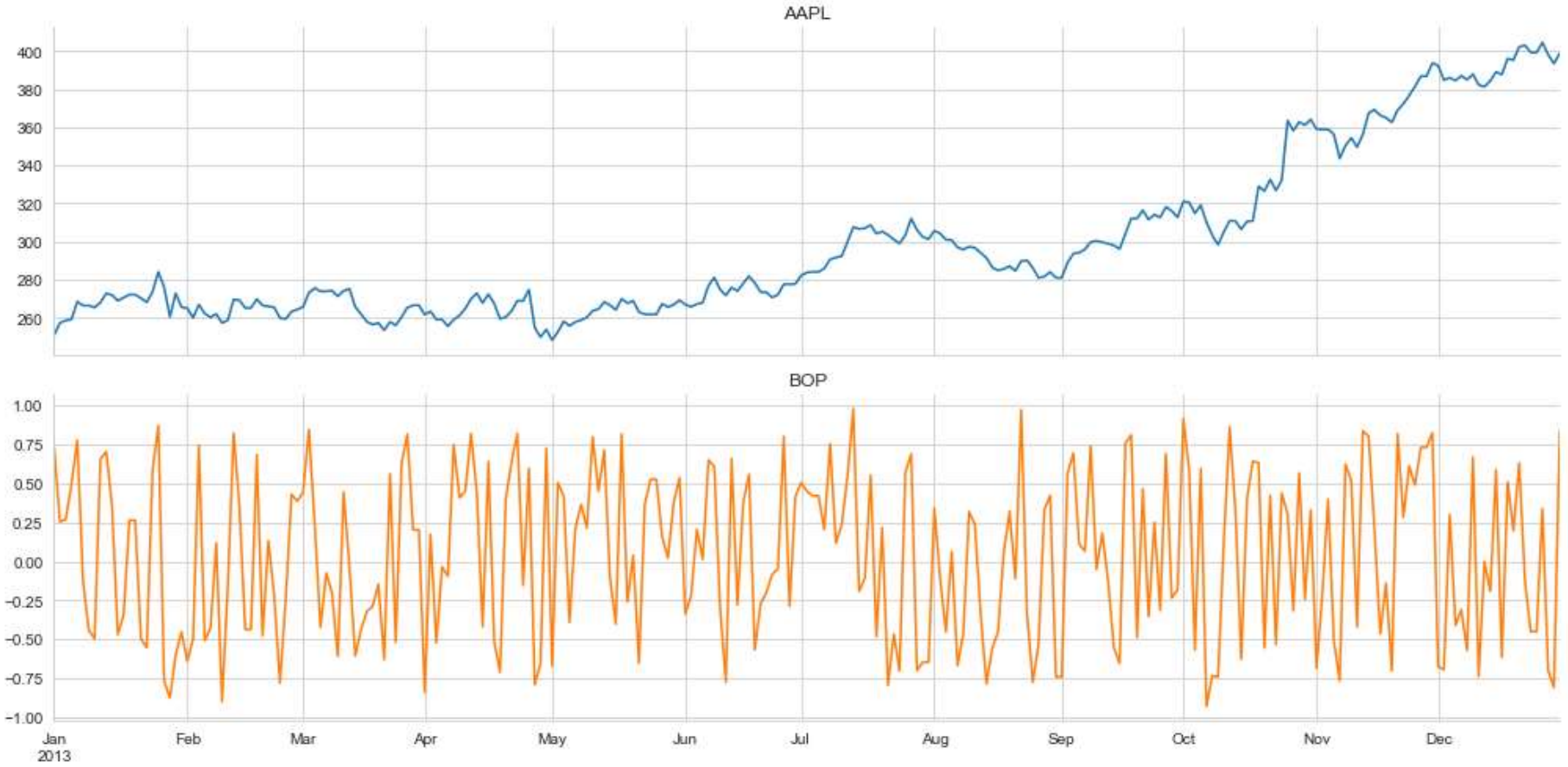


Balance Of Power (BOP)

In [86]:
```
df = price_sample.loc['2013', ['open', 'high', 'low', 'close']]
```

In [87]:
```
df['BOP'] = talib.BOP(open=df.open,
                      high=df.high,
                      low=df.low,
                      close=df.close)
```

In [88]:
```
axes = df[['close', 'BOP']].plot(figsize=(14, 7), rot=0, subplots=True, title=['AAPL', 'BOP'], legend=False)
axes[1].set_xlabel('')
sns.despine()
plt.tight_layout();
```



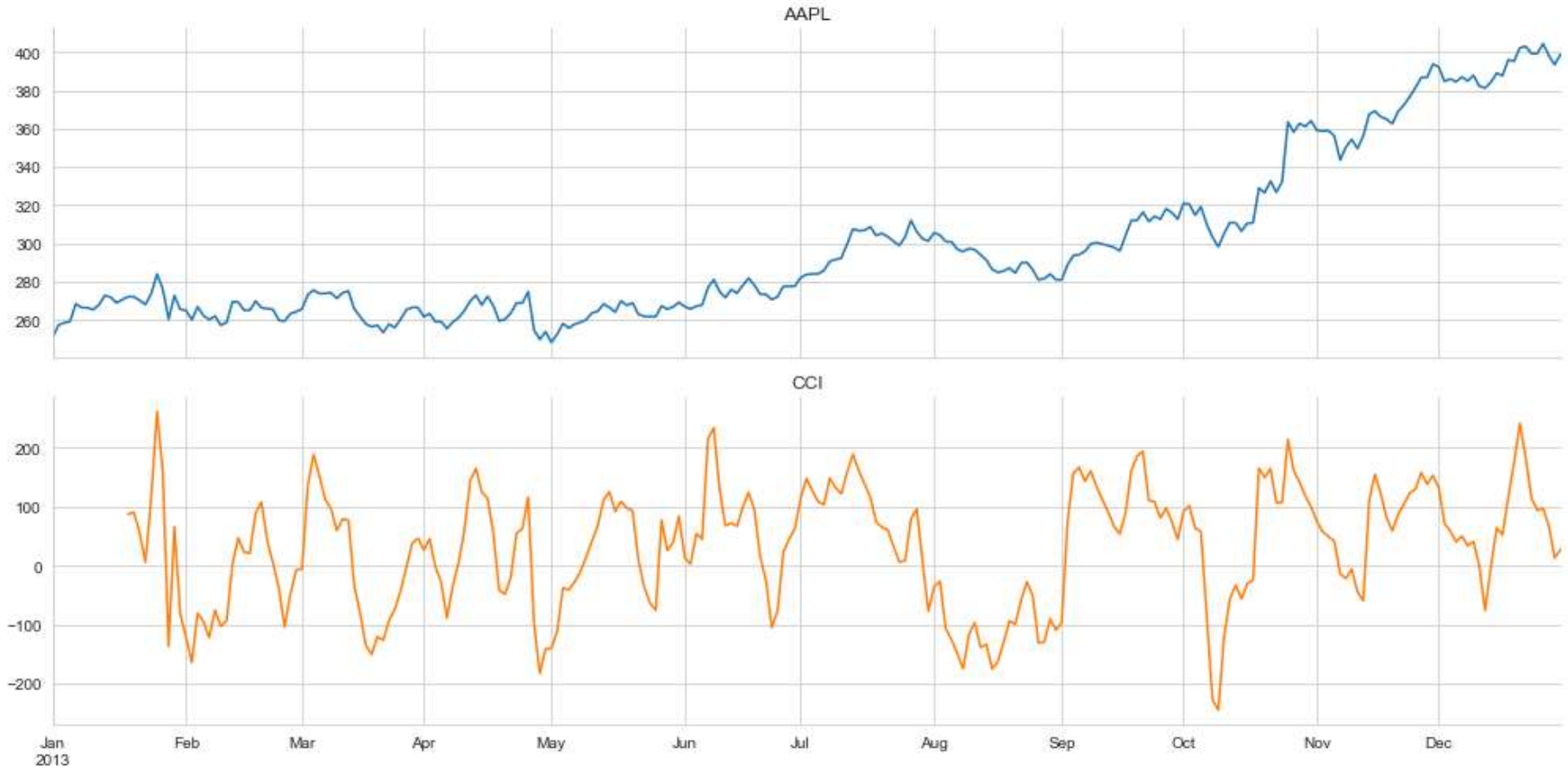
Commodity Channel Index (CCI)

In [92]:
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [93]:
```
df['CCI'] = talib.CCI(high=df.high,
                      low=df.low,
                      close=df.close,
                      timeperiod=14)
```

In [94]:
```
axes = df[['close', 'CCI']].plot(figsize=(14, 7),
                                 rot=0,
```

```
                                subplots=True,
                                title=['AAPL', 'CCI'],
                                legend=False)
axes[1].set_xlabel('')
sns.despine()
plt.tight_layout()
```



Moving Average Convergence/Divergence (MACD)

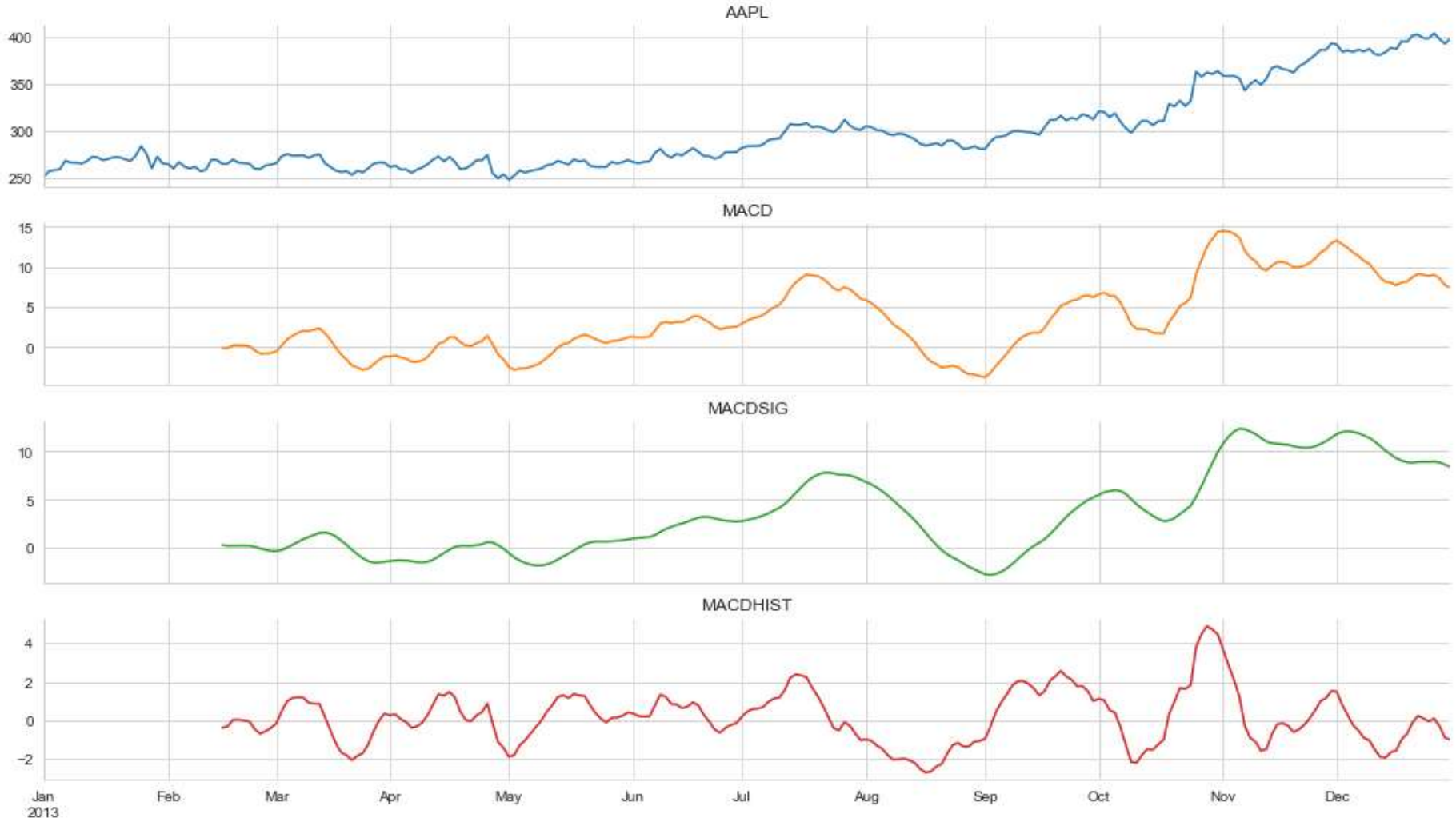
In [97]:
```
df = price_sample.loc['2013', ['close']]
```

In [98]:
```
macd, macdsignal, macdhist = talib.MACD(df.close,
```

```
                                        fastperiod=12,
                                        slowperiod=26,
                                        signalperiod=9)
df['MACD'] = macd
df['MACDSIG'] = macdsignal
df['MACDHIST'] = macdhist
```

In [99]:

```
axes = df.plot(figsize=(14, 8),
               rot=0,
               subplots=True,
               title=['AAPL', 'MACD', 'MACDSIG', 'MACDHIST'],
               legend=False)

axes[-1].set_xlabel('')
sns.despine()
plt.tight_layout()
```



Chande Momentum Oscillator (CMO)

In [105…
```
df = price_sample.loc['2013', ['close']]
```

In [106…
```
df['CMO'] = talib.CMO(df.close, timeperiod=14)
```

In [107…
```
ax = df.plot(figsize=(14, 4), rot=0, secondary_y=['CMO'], style=['-', '--'])
```

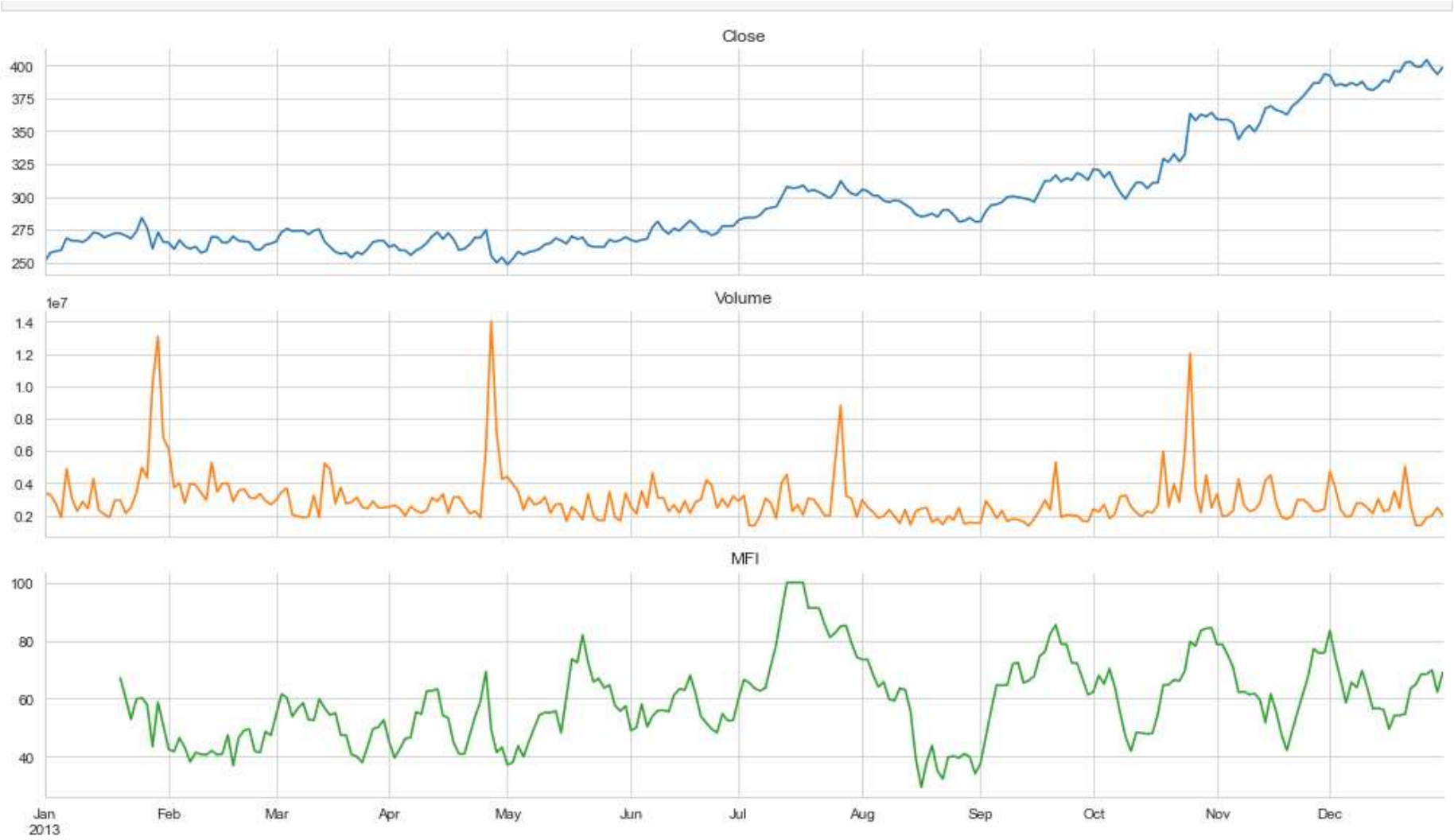
```
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

Money Flow Index

In [110…
```
df = price_sample.loc['2013', ['high', 'low', 'close', 'volume']]
```

In [111…
```
df['MFI'] = talib.MFI(df.high,
                      df.low,
                      df.close,
                      df.volume,
                      timeperiod=14)
```

In [112…
```
axes = df[['close', 'volume', 'MFI']].plot(figsize=(14, 8),
                                           rot=0,
                                           subplots=True,
                                           title=['Close', 'Volume', 'MFI'],
                                           legend=False)
axes[-1].set_xlabel('')
sns.despine()
plt.tight_layout()
```



Relative Strength Index

In [115…
```
df = price_sample.loc['2013', ['close']]
```

In [116…
```
df['RSI'] = talib.RSI(df.close, timeperiod=14)
```

In [117…
```
ax = df.plot(figsize=(14, 4), rot=0, secondary_y=['RSI'], style=['-', '--'])
```

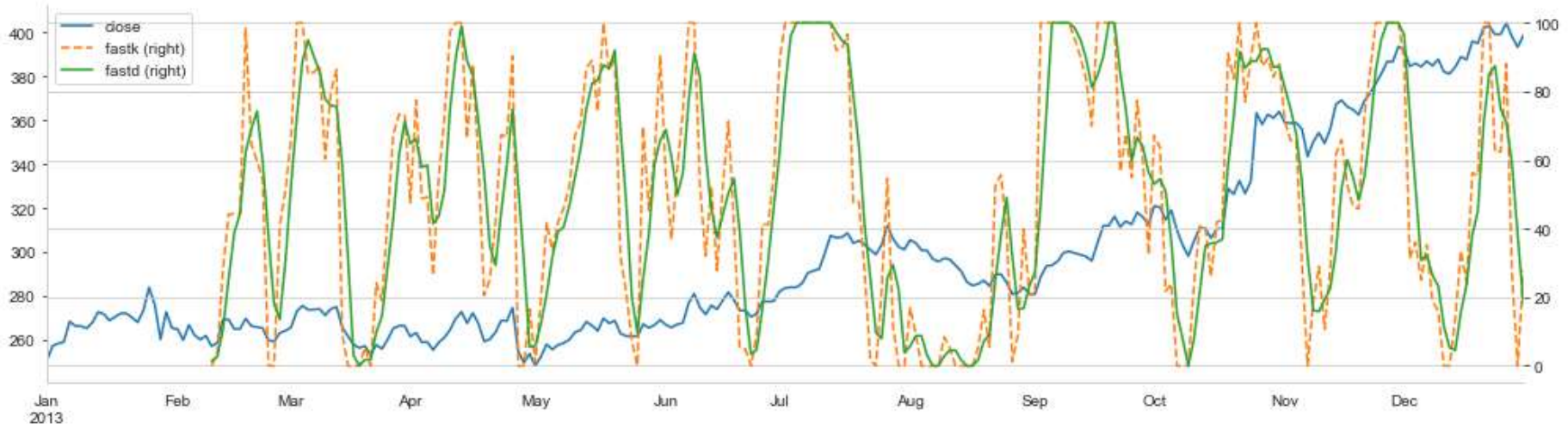
```
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

Stochastic RSI (STOCHRSI)

In [120…
```
df = price_sample.loc['2013', ['close']]
```

In [121…
```
fastk, fastd = talib.STOCHRSI(df.close,
                              timeperiod=14,
                              fastk_period=14,
                              fastd_period=3,
                              fastd_matype=0)
df['fastk'] = fastk
df['fastd'] = fastd
```

In [122…
```
ax = df.plot(figsize=(14, 4),
             rot=0,
             secondary_y=['fastk', 'fastd'], style=['-', '--'])
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

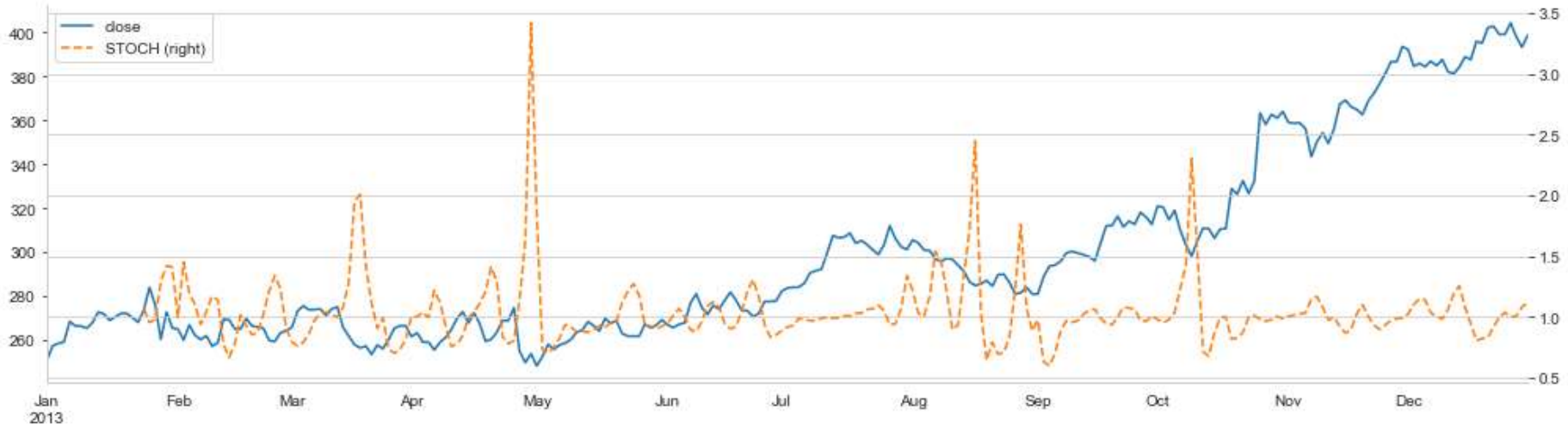


Stochastic (STOCH)

In [124…
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [125…
```
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
```

In [126…
```
ax = df[['close', 'STOCH']].plot(figsize=(14, 4),
                                 rot=0,
                                 secondary_y='STOCH', style=['-', '--'])
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

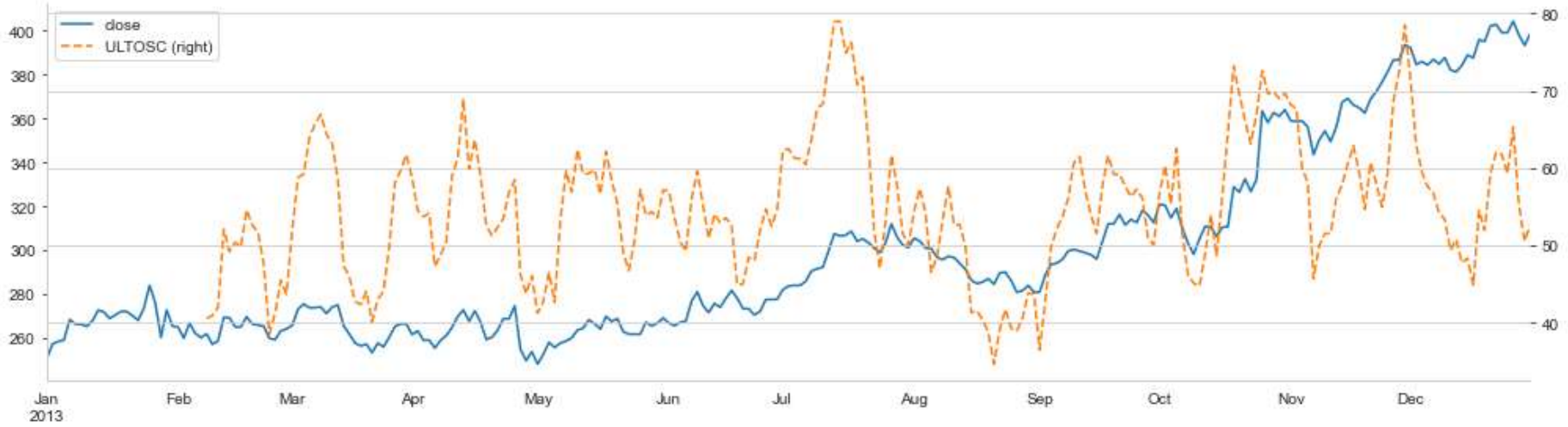


Ultimate Oscillator (ULTOSC)

In [130…
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [131…
```
df['ULTOSC'] = talib.ULTOSC(df.high,
                            df.low,
                            df.close,
                            timeperiod1=7,
                            timeperiod2=14,
                            timeperiod3=28)
```

In [132…
```
ax = df[['close', 'ULTOSC']].plot(figsize=(14, 4),
                                  rot=0,
                                  secondary_y='ULTOSC', style=['-', '--'])
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

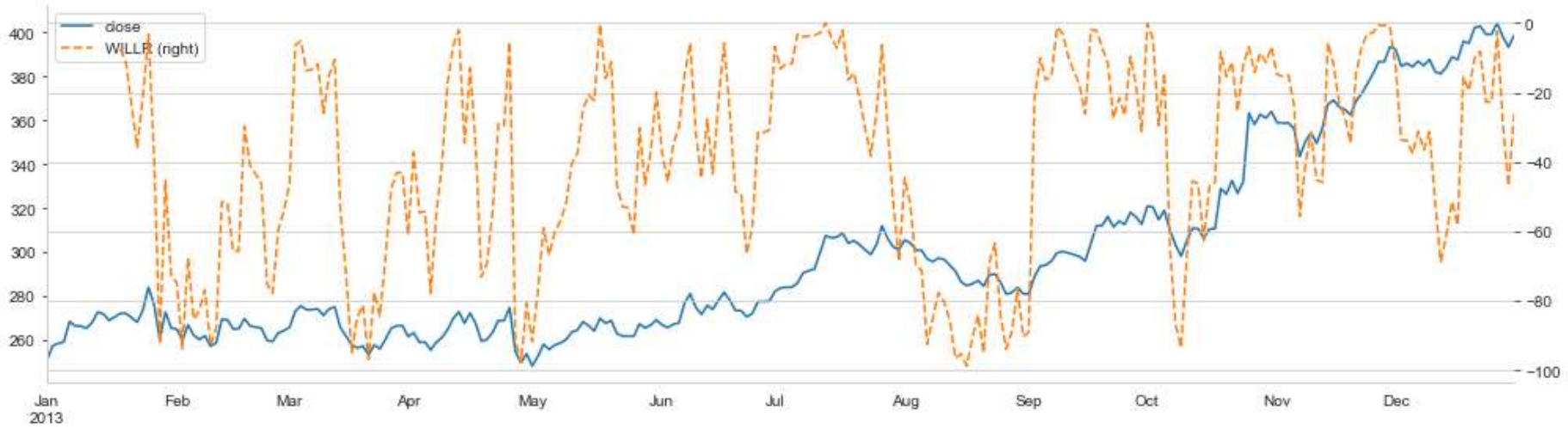


Williams' %R (WILLR)

In [136…
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [137…
```
df['WILLR'] = talib.WILLR(df.high,
                          df.low,
                          df.close,
                          timeperiod=14)
```

In [138…
```
ax = df[['close', 'WILLR']].plot(figsize=(14, 4),
                                 rot=0,
                                 secondary_y='WILLR', style=['-', '--'])
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```



Volume Indicators

In [141…
```
df = price_sample.loc['2013', ['high', 'low', 'close', 'volume']]
```

In [142…
```
df['AD'] = talib.AD(df.high,
                    df.low,
                    df.close,
                    df.volume)
```

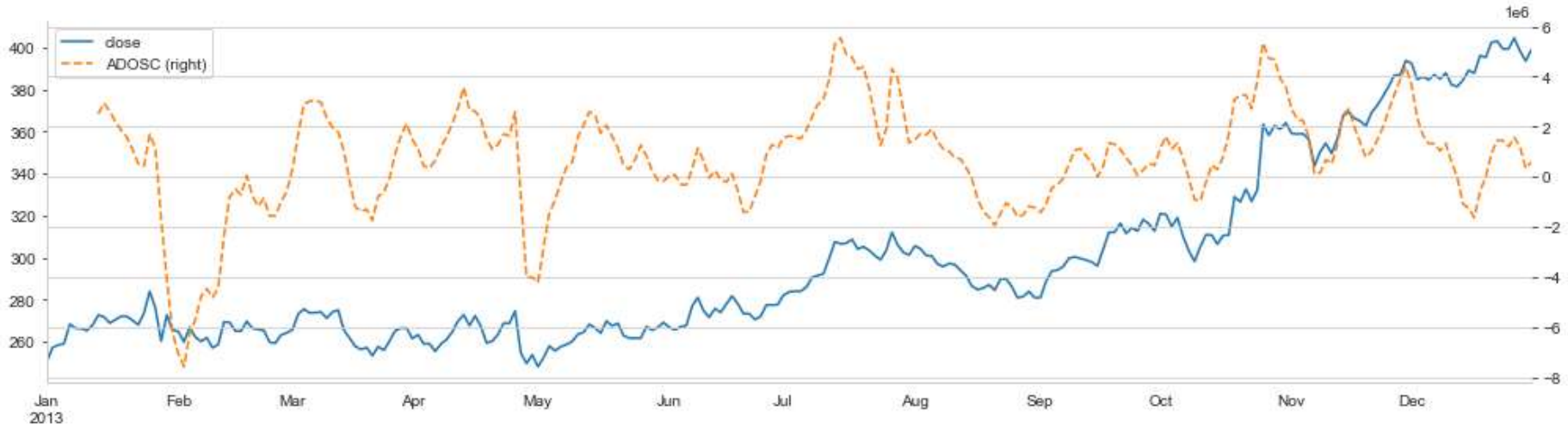
In [143…
```
ax = df[['close', 'AD']].plot(figsize=(14, 4),
                              rot=0,
                              secondary_y='AD', style=['-', '--'])

ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

Chaikin A/D Oscillator (ADOSC)

In [147…
```
df = price_sample.loc['2013', ['high', 'low', 'close', 'volume']]
```

In [148…
```
df['ADOSC'] = talib.ADOSC(df.high,
                          df.low,
                          df.close,
                          df.volume,
                          fastperiod=3,
                          slowperiod=10)
```

In [149…
```
ax = df[['close', 'ADOSC']].plot(figsize=(14, 4),
                                 rot=0,
                                 secondary_y='ADOSC', style=['-', '--'])

ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```



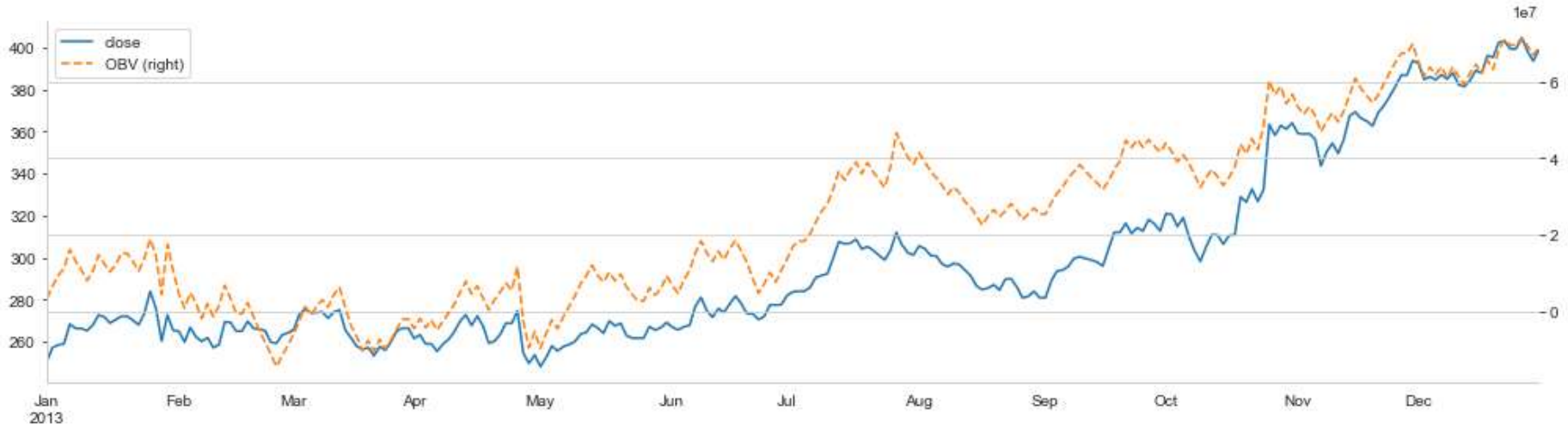
In [150…
```
data['ADOSC'] = by_ticker.apply(lambda x: talib.ADOSC(x.high,
                                                      x.low,
                                                      x.close,
                                                      x.volume,
                                                      fastperiod=3,
                                                      slowperiod=10)/x.rolling(14).volume.mean())
```

On Balance Volume (OBV)

In [152…
```
df = price_sample.loc['2013', ['close', 'volume']]
```

In [153…
```
df['OBV'] = talib.OBV(df.close,
                      df.volume)
```

In [154…
```
ax = df[['close', 'OBV']].plot(figsize=(14, 4),
                               rot=0,
                               secondary_y='OBV', style=['-', '--'])
ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```



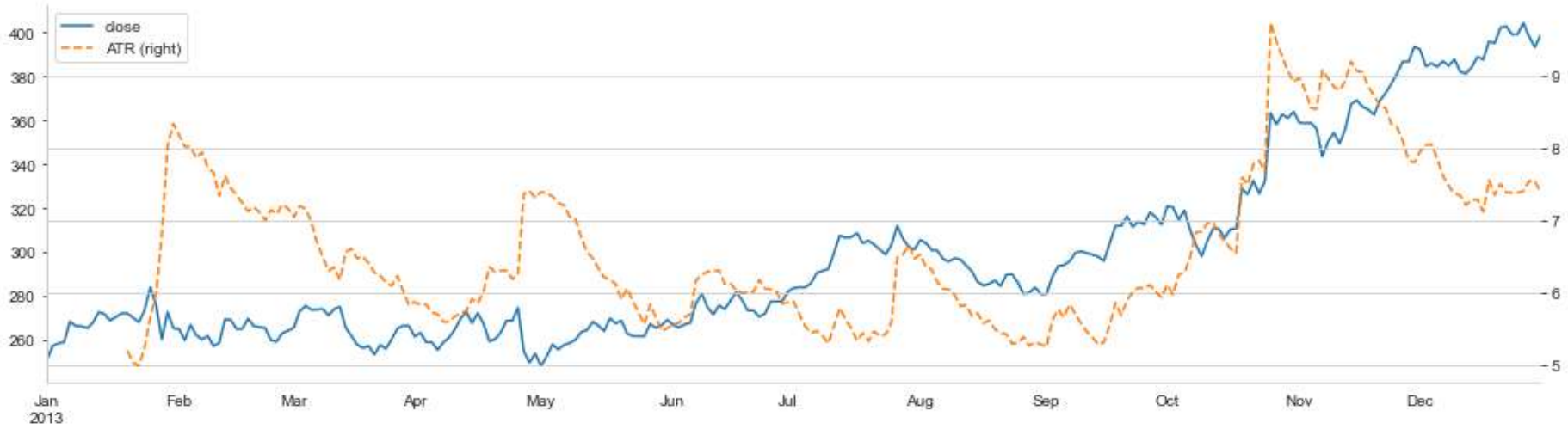
In [155…
```
data['OBV'] = by_ticker.apply(lambda x: talib.OBV(x.close,
                                                  x.volume)/x.expanding().volume.mean())
```

Volatility Indicators

In [157…
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [158…
```
df['ATR'] = talib.ATR(df.high,
                      df.low,
                      df.close,
                      timeperiod=14)
```

In [159…
```
ax = df[['close', 'ATR']].plot(figsize=(14, 4),
                               rot=0,
                               secondary_y='ATR', style=['-', '--'])

ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```

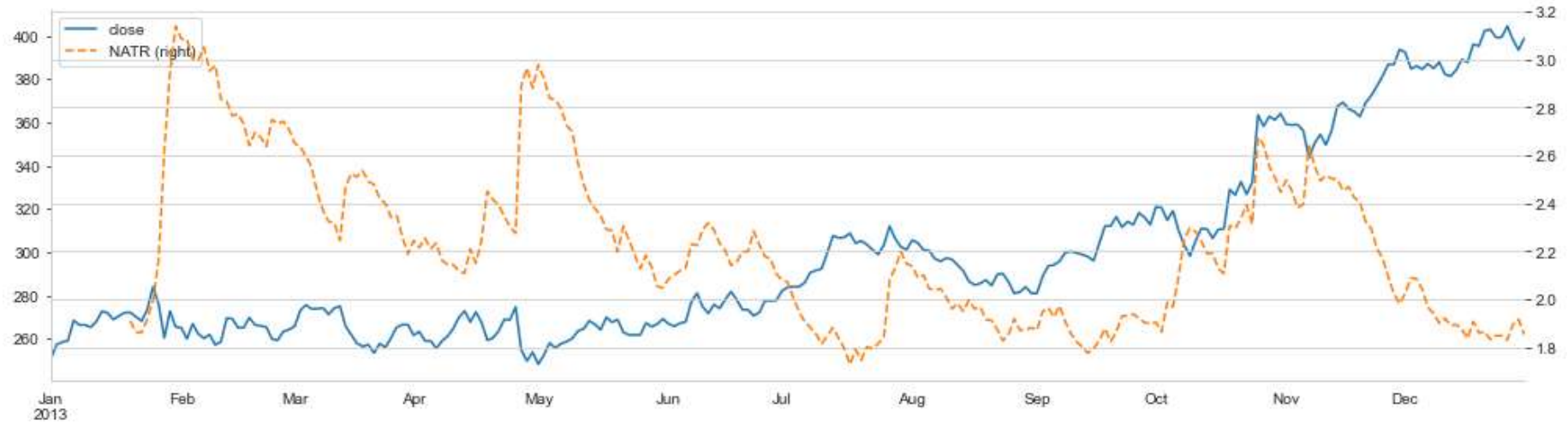


NATR

In [162…
```
df = price_sample.loc['2013', ['high', 'low', 'close']]
```

In [163…
```
df['NATR'] = talib.NATR(df.high,
                        df.low,
                        df.close,
                        timeperiod=14)
```

In [164…
```
ax = df[['close', 'NATR']].plot(figsize=(14, 4),
                                rot=0,
                                secondary_y='NATR', style=['-', '--'])

ax.set_xlabel('')
sns.despine()
plt.tight_layout()
```



In [171…
```
data = (data
        .drop(['open', 'high', 'low', 'close', 'volume'], axis=1)
        .replace((np.inf, -np.inf), np.nan))
```

In [172…
```
data.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 1403584 entries, ('A', Timestamp('2006-01-03 00:00:00')) to ('ZMH', Timestamp('2015-07-02 00:00:00'))
Data columns (total 55 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   ex-dividend   1403584 non-null  float64
 1   split_ratio   1403584 non-null  float64
 2   adj_open      1403584 non-null  float64
 3   adj_high      1403584 non-null  float64
 4   adj_low       1403584 non-null  float64
 5   adj_close     1403584 non-null  float64
 6   adj_volume    1403584 non-null  float64
 7   ret_01        1403084 non-null  float64
 8   ret_03        1402084 non-null  float64
 9   ret_05        1401084 non-null  float64
 10  ret_10        1398584 non-null  float64
 11  ret_21        1393084 non-null  float64
 12  ret_42        1382584 non-null  float64
 13  ret_63        1372084 non-null  float64
 14  ret_126       1340584 non-null  float64
 15  ret_252       1277584 non-null  float64
 16  ret_fwd       1403584 non-null  float64
 17  BB_UP         1394084 non-null  float64
 18  BB_LOW        1394084 non-null  float64
 19  BB_SQUEEZE    1394084 non-null  float64
 20  HT            1372084 non-null  float64
 21  SAR           1403084 non-null  float64
 22  ADX           1390084 non-null  float64
 23  ADXR          1383584 non-null  float64
 24  PPO           1391084 non-null  float64
 25  AARONOSC      1396584 non-null  float64
 26  BOP           1403584 non-null  float64
 27  CCI           1397084 non-null  float64
 28  MACD          1387084 non-null  float64
 29  MACD_SIGNAL   1387084 non-null  float64
 30  MACD_HIST     1387084 non-null  float64
 31  MFI           1396584 non-null  float64
 32  RSI           1396584 non-null  float64
 33  STOCHRSI      1389084 non-null  float64
 34  STOCH         1395066 non-null  float64
```

```
 35  ULTOSC       1389584 non-null  float64
 36  WILLR        1397084 non-null  float64
 37  AD           1403584 non-null  float64
 38  ADOSC        1396996 non-null  float64
 39  OBV          1403583 non-null  float64
 40  ATR          1396584 non-null  float64
 41  ALPHA_63     1334328 non-null  float64
 42  MARKET_63    1334328 non-null  float64
 43  SMB_63       1334328 non-null  float64
 44  HML_63       1334328 non-null  float64
 45  RMW_63       1334328 non-null  float64
 46  CMA_63       1334328 non-null  float64
 47  ALPHA_252    1239828 non-null  float64
 48  MARKET_252   1239828 non-null  float64
 49  SMB_252      1239828 non-null  float64
 50  HML_252      1239828 non-null  float64
 51  RMW_252      1239828 non-null  float64
 52  CMA_252      1239828 non-null  float64
 53  month        1403584 non-null  uint8
 54  weekday      1403584 non-null  uint8
dtypes: float64(53), uint8(2)
memory usage: 608.1+ MB
```