Sarit Maitra

ALGORITHMIC TRADING SIMPLIFIED

Evaluate Price Oscillator Stock Trading Strategy

Algorithmic Trading and PnL Evaluation

Sarit Maitra Apr 14 · 10 min read ★

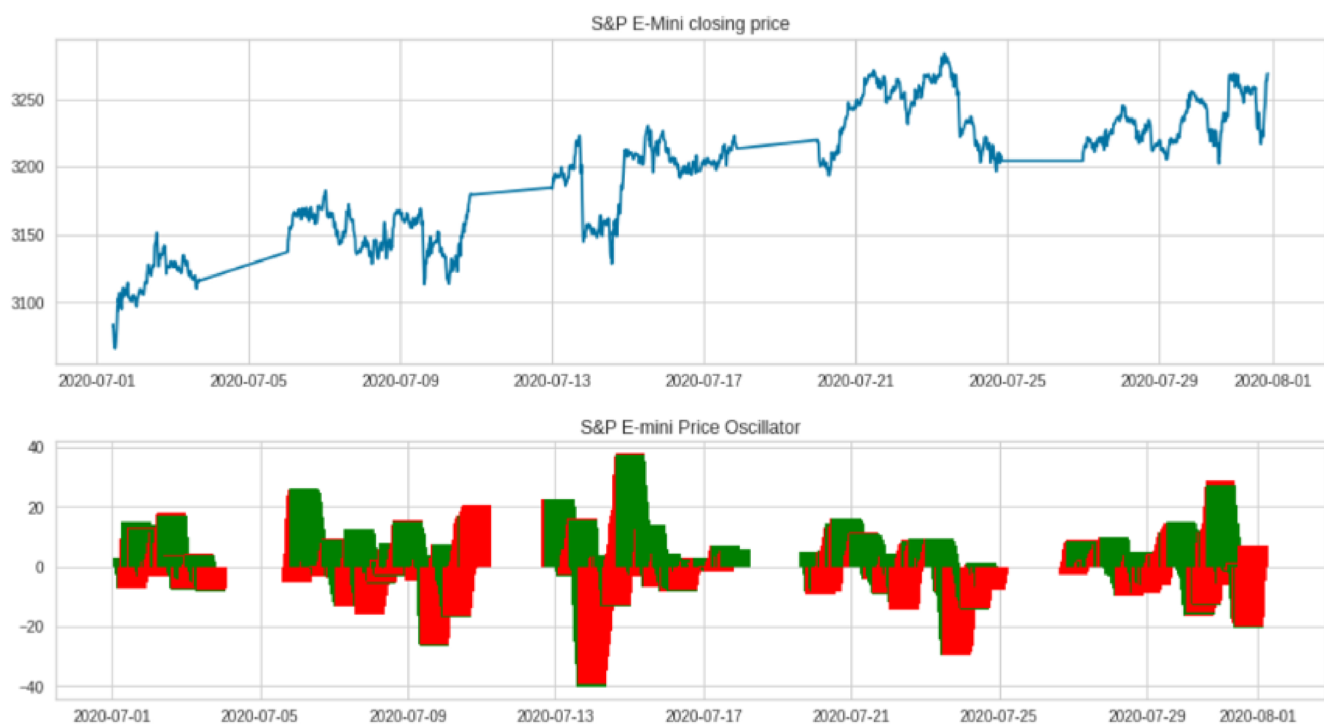


Image by Author

Price Oscillator is a technical indicator calculating the difference between two price moving averages. It is quite similar to MACD indicator. Crossovers of two moving averages correspond to crossovers of price oscillator [PO (MACD)] and zero central signal line around it oscillates.

To understand price oscillator, we need to know the exponential moving averages (EMA) concept. In simple term, EMA is the average price over a certain number of days, with more recent days weighted more heavily i.e. exponentially. EMA needs to be calculated over a period of appropriate length to maximize meaningful data while minimizing random movement. There are some basic thumb rule as 26 days for long EMA and 13 days (well between 9–13 days) for short EMA. However, these are arbitrary and can be adjusted based on strategic movement.

Now, price oscillator is: (ema(short) — ema(long)) / ema(long). So, essentially, we're looking at the short-term average as a fraction of the long-term average; hence, price oscillator.

Also, because the price oscillator compares two EMAs, we are able to compare movements through different time frames. Here, we will experiment with absolute price oscillator and develop a PnL program.

We will experiment with SP e-mini futures index with 15 min frequency data set.

```
def SPFutures():
  df = pd.read_csv("July_Futures_15min.csv")
  df.set_index('timestamp', inplace=True)
  df.sort_index(ascending=True, inplace=True)
  df.index = pd.to_datetime(df.index)
  df = df[df['volume'] > 0]
  df.sort_index(ascending=True, inplace=True)
  df.drop_duplicates(inplace=True) # dropping duplicates if any
  df.fillna(method='pad', inplace=True)
  return (df)

df = SPFutures()
print(df.head())
```

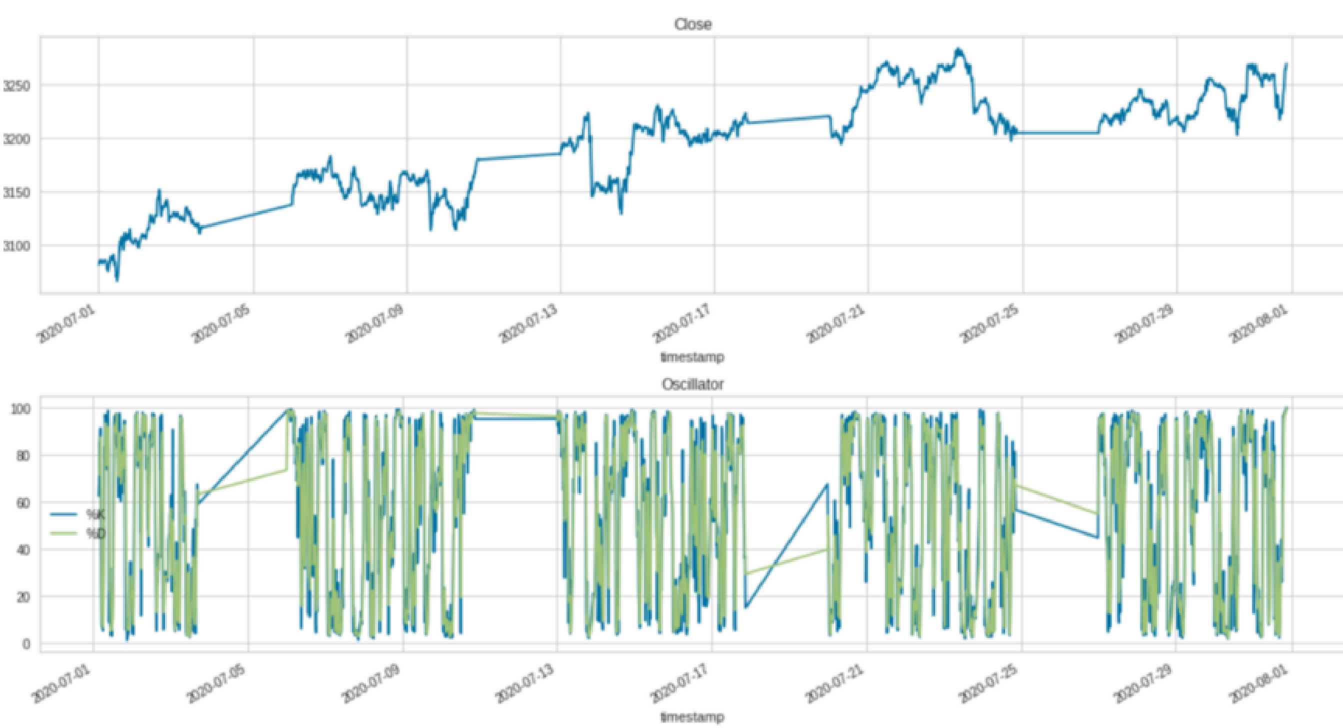
```
                        es_close      es_open      es_high       es_low      volume
timestamp
2020-07-01 00:00:00  3081.169643  3081.116071  3081.660714  3080.625000  147.857143
2020-07-01 00:15:00  3081.633333  3081.533333  3082.000000  3081.158333   72.133333
2020-07-01 00:30:00  3084.941667  3084.741667  3085.308333  3084.391667   85.733333
2020-07-01 00:45:00  3084.750000  3084.850000  3085.191667  3084.366667   69.400000
```

```
2020-07-01 01:00:00  3085.016667  3084.991667  3085.541667  3084.425000   96.133333
```

Stochastic Oscillator:

Let us first see how a simple Stochastic Oscillator based strategy works with the given data set and plot an equity curve to display cumulative returns.

```
d = df.copy()

# Create the "L14" column in the DataFrame
d['L_14'] = d['es_low'].rolling(window=14).min()
#Create the "H14" column in the DataFrame
d['H_14'] = d['es_high'].rolling(window=14).max()
#Create the "%K" column in the DataFramed['%K'] = 100*((d['es_close']
- d['L_14']) / (d['H_14'] - d['L_14']))
#Create the "%D" column in the DataFrame
d['%D'] = d['%K'].rolling(window=3).mean()

#  plot
fig, axes = plt.subplots(nrows=2, ncols=1,figsize=(15,10))
d['es_close'].plot(ax=axes[0]); axes[0].set_title('Close')
d[['%K','%D']].plot(ax=axes[1]); axes[1].set_title('Oscillator')
plt.tight_layout(); plt.show()
```



Let us run a simple back test and draw an equity curve to check the performance of the strategy. Here, our strategy is:

- a sell is initiated when the %K line crosses down through the %D line and the value of the oscillator is above 75.
- exit from sell signal is given when the %K line crosses back up through the %D line

Likewise:

- buy is initiated when the %K line crosses up through the %D line and the value of the oscillator is below 25
- exit from buy signal is given when the %K line crosses back down through the %D line

Let us back-test the goodness-of-fit of our strategy.

```
d['enter_sell'] = ((d['%K'] < d['%D']) & (d['%K'].shift(1) >
d['%D'].shift(1))) & (d['%D'] > 75)

d['exit_sell'] = ((d['%K'] > d['%D']) & (d['%K'].shift(1) <
d['%D'].shift(1)))

d['short'] = np.nan
d.loc[d['enter_sell'],'short'] = -1
d.loc[d['exit_sell'],'short'] = 0

#Set initial position on day 1 to flat
d['short'][0] = 0
d['short'] = d[short'].fillna(method='pad')

d['enter_buy'] = ((d['%K'] > d['%D']) & (d['%K'].shift(1) <
d['%D'].shift(1))) & (d['%D'] < 25)

d['exit_buy'] = ((d['%K'] < d['%D']) & (d['%K'].shift(1) >
d['%D'].shift(1)))

d['long'] = np.nan
d.loc[d['enter_buy'],'long'] = 1
d.loc[d['exit_buy'],'long'] = 0

d['long'][0] = 0
d['long'] = d['long'].fillna(method='pad')
```

```
# Add Long and Short positions together to get final strategy
position (1 for long, -1 for short and 0 for flat)

d['Position'] = d['long'] + d['short']
```

Visualization:

```
d['market_returns'] = d['es_close'].pct_change()
d['strategy_returns'] = d['market_returns'] * d['Position'].shift(1)
d[['strategy_returns','market_returns']].cumsum().plot()
plt.show()
```

So, here we have experimented with Stochastic Oscillator (SO) to find that out that, simple SO based trading strategy shows positive return for the month. However, looking at the market return plot, a simple buy and hold strategy would have been profitable too.

Let us experiment with absolute price oscillator on the same data set and explore more details by doing a back test. As discussed earlier, APO needs fast and slow moving EMAs to go forward.

Calculating Price Oscillator:

The Price Oscillator is simply the nine-day EMA, decreased and then divided by the 26-day EMA {[EMA(9day) — EMA(26day)] / EMA(26day)}.

> Tradition and convention have deemed 26 days to be the dividing line between the short term and the "minor intermediate" term in the stock market, with the "very short" term lasting between five and 13 days.(Investopedia)

Parameters for EMA Calculation:

Let us consider 10 (fast EMA) and 40 (slow ema) parameters.

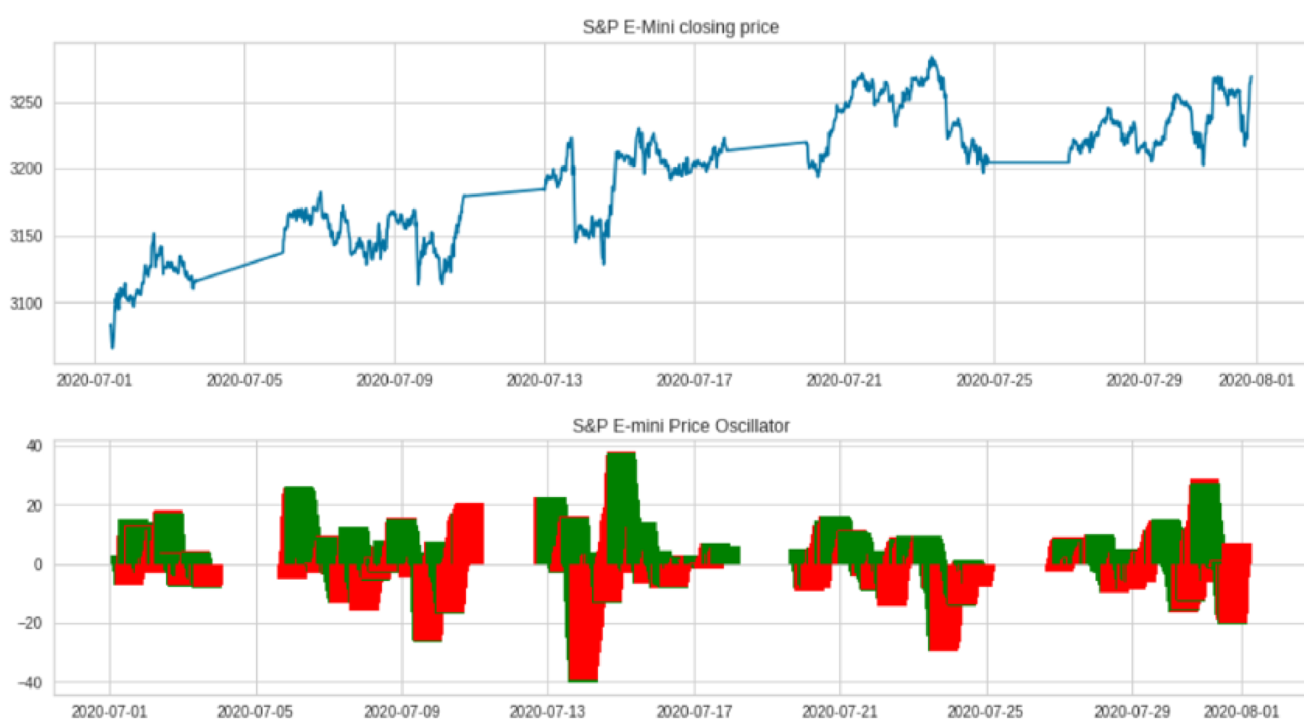
When we calculate MA, we sum up the closing prices over the past n-times and then divide by n. The EMA calculation uses a smoothing factor to place a higher weight on recent data points and is considered as much more efficient than the linear weighted average.

1. We calculate the smoothing factor: multiplier = 2 / (EMA length + 1).
2. EMA[0] = (close[0] − EMA[1]) * multiplier + EMA[1].

```
fastPeriod = 10
fastSmooth = 2 / (fastPeriod + 1)
fastEma = 0
fastEma_val = []

slowPeriod = 40
slowSmooth = 2 / (slowPeriod + 1)
slowEma = 0
slowEma_val = []
poValues = []
```

We are also storing the price oscillator values to track computed Price Oscillator value signals. Let us calculate the values of the Price Oscillator and visualize the histogram.

```
def ema(price, period):
  ema = price.rolling(period).mean()
  return ema

def po(price, period1, period2):
  median = price.rolling(2).median()
  short = ema(median, period1)
  long = ema(median, period2)
  po = short - long
  po_df = DataFrame(po).rename(columns = {'Close':'po'})
  return po_df

df['po'] = po(df['es_close'], 10, 40)
df.dropna(inplace=True)
plt.figure(figsize=(15,8))

ax1 = plt.subplot2grid((10,1), (0,0), rowspan = 5, colspan = 1)
ax2 = plt.subplot2grid((10,1), (6,0), rowspan = 4, colspan = 1)
ax1.plot(df['es_close'])
ax1.set_title('S&P E-Mini closing price')

for i in range(len(df)):
  if df['po'][i-1] > df['po'][i]:
    ax2.bar(df.index[i], df['po'][i], color = 'green')
  else:
    ax2.bar(df.index[i], df['po'][i], color = 'red')

ax2.set_title('S&P E-mini Price Oscillator')
plt.show()
```



We can see, whenever the current bar > previous bar, green bar is plotted on the chart, and, whenever the current bar < previous bar, red bar is plotted on the chart.

Trading strategy:

Let us create variables for trading strategy, position & pnl management.

- orders [] and positions[] are two containers we have created to track buy/sell order, +1 for buy order, -1 for sell order, 0 for no-action and track positions, +ve for long positions, -ve for short positions, 0 for flat/no position respectively.
- pnls[] for the sum of closed_pnl i.e. pnls already locked in and open_pnl i.e. pnls for open-position marked to market price
- last_buyPrice and last_sellPrice are the prices at which last buy or sell was made; these will be used to prevent over-trading at/around the same price.
- position is the current position of the trading strategy
- buy_sumPrice & sell_sumPrice are the summation of products of buy_trade_price and buy_trade_qty for every buy Trade made since last time being flat

```
orders = []
positions = []
pnls = []
last_buyPrice = 0
last_sellPrice = 0
position = 0
buy_sumPrice_qty = 0
buy_sum_qty = 0
sell_sumPrice_qty = 0
sell_sum_qty = 0
open_pnl = 0
closed_pnl = 0
```

Let is define some constants to define strategy behavior/thresholds.

```
po_buyEnter = -10
po_sellEnter = 10

min_price_movement_from_last_trade = 10
num_stocks_per_trade = 10
```

```
min_profit_to_close = 10 * num_stocks_per_trade
```

Let us define constants/variables that are used to compute standard deviation as a volatility measure.

Here our strategy is simple:

We will perform a sell trade at close_price if the following conditions are met:

- The PO trading signal value is above Sell-Entry threshold and the difference between last trade-price and current-price is different enough.
- We are long(+ve position) and either APO trading signal value is at or above 0 or current position is profitable enough to lock profit.

We will perform a buy trade at close_price if the following conditions are met:

- The PO trading signal value is below Buy-Entry threshold and the difference between last trade-price and current-price is different enough.
- We are short(-ve position) and either APO trading signal value is at or below 0 or current position is profitable enough to lock profit.

```
maPeriods = 20
priceHist = []

close = df['es_close'].copy()

for close_price in close:
  priceHist.append(close_price)
  if len(priceHist) > maPeriods:
    del (priceHist[0])

  sma = stats.mean(priceHist)
  variance = 0
  for histPrice in priceHist:
    variance = variance + ((histPrice - sma) ** 2)

  stdev = math.sqrt(variance / len(priceHist))
  stdev_factor = stdev/15
  if stdev_factor == 0:
    stdev_factor = 1
```

```
  if (fastEma == 0):
    fastEma = close_price
    slowEma = close_price
  else:
    fastEma = (close_price - fastEma) * smooth_fast * stdev_factor +
fastEma
    slowEma = (close_price - slowEma) * smooth_slow * stdev_factor +
slowEma

  fastEma_val.append(fastEma)
  slowEma_val.append(slowEma)

  po = fastEma - slowEma
  poValues.append(po)

  if ((po > po_sellEnter * stdev_factor and abs(close_price -
last_sellPrice) > min_price_movement_from_last_trade * stdev_factor)

  or
  (position > 0 and (po >= 0 or open_pnl > min_profit_to_close /
stdev_factor))):
  orders.append(-1) # sell trade
  last_sellPrice = close_price
  position -= num_stocks_per_trade
  sell_sumPrice_qty += (close_price * num_stocks_per_trade) # update
vwap sell-price
  sell_sum_qty += num_stocks_per_trade
  print( "SELL ", num_stocks_per_trade, " @ ", close_price, "Position
=> ", position)

  elif ((po < po_buyEnter * stdev_factor and abs(close_price -
last_buyPrice) > min_price_movement_from_last_trade * stdev_factor)
  or
  (position < 0 and (po <= 0 or open_pnl > min_profit_to_close /
stdev_factor))):
  orders.append(+1) # buy trade
  last_buyPrice = close_price
  position += num_stocks_per_trade # increase position by the size of
this trade

  buy_sumPrice_qty += (close_price * num_stocks_per_trade) # update
the vwap buy-price
  buy_sum_qty += num_stocks_per_trade
  print( "BUY ", num_stocks_per_trade, " @ ", close_price, "POSITION:
", position ); print()
  else:
  orders.append(0)
  positions.append(position)

  open_pnl = 0
  if position > 0:
    if sell_sum_qty > 0:
```

```
      open_pnl = abs(sell_sum_qty) * (sell_sumPrice_qty/sell_sum_qty
- buy_sumPrice_qty/buy_sum_qty)

    open_pnl += abs(sell_sum_qty - position) * (close_price -
buy_sumPrice_qty / buy_sum_qty)
  elif position < 0:
    if buy_sum_qty > 0:
      open_pnl = abs(buy_sum_qty) * (sell_sumPrice_qty/sell_sum_qty -
buy_sumPrice_qty/buy_sum_qty)

    open_pnl += abs(buy_sum_qty - position) *
(sell_sumPrice_qty/sell_sum_qty - close_price)

  else:
    closed_pnl += (sell_sumPrice_qty - buy_sumPrice_qty)
    buy_sumPrice_qty = 0
    buy_sum_qty = 0
    sell_sumPrice_qty = 0
    sell_sum_qty = 0
    last_buyPrice = 0
    last_sellPrice = 0

  print( "Open/PnL-> ", open_pnl, " Closed/PnL-> ", closed_pnl, "
Total/PnL-> ", (open_pnl + closed_pnl) );
  pnls.append(closed_pnl + open_pnl)
```

```
Open/PnL->  10656.333333333187  Closed/PnL->  56315.41666666452  Total/PnL->  66971.7499999977
BUY  10  @  3222.258333333333 POSITION:  -10

Open/PnL->  11177.708333333121  Closed/PnL->  56315.41666666452  Total/PnL->  67493.12499999764
BUY  10  @  3227.575 POSITION:  0

Open/PnL->  0  Closed/PnL->  61333.83333333103  Total/PnL->  61333.83333333103
BUY  10  @  3234.675 POSITION:  10

Open/PnL->  0.0  Closed/PnL->  61333.83333333103  Total/PnL->  61333.83333333103
BUY  10  @  3240.808333333333 POSITION:  20

Open/PnL->  61.33333333333212  Closed/PnL->  61333.83333333103  Total/PnL->  61395.1666666436
Open/PnL->  130.66666666667516  Closed/PnL->  61333.83333333103  Total/PnL->  61464.4999999977
BUY  10  @  3246.766666666667 POSITION:  30

Open/PnL->  180.50000000000637  Closed/PnL->  61333.83333333103  Total/PnL->  61514.33333333104
SELL  10  @  3254.6583333333333 Position =>  20
Open/PnL->  278.1666666666606  Closed/PnL->  61333.83333333103  Total/PnL->  61611.99999999769
SELL  10  @  3263.5916666666667 Position =>  10
Open/PnL->  595.916666666667  Closed/PnL->  61333.83333333103  Total/PnL->  61929.74999999769
SELL  10  @  3264.1 Position =>  0
Open/PnL->  0  Closed/PnL->  61934.83333333103  Total/PnL->  61934.83333333103
Open/PnL->  0  Closed/PnL->  61934.83333333103  Total/PnL->  61934.83333333103
Open/PnL->  0  Closed/PnL->  61934.83333333103  Total/PnL->  61934.83333333103
```

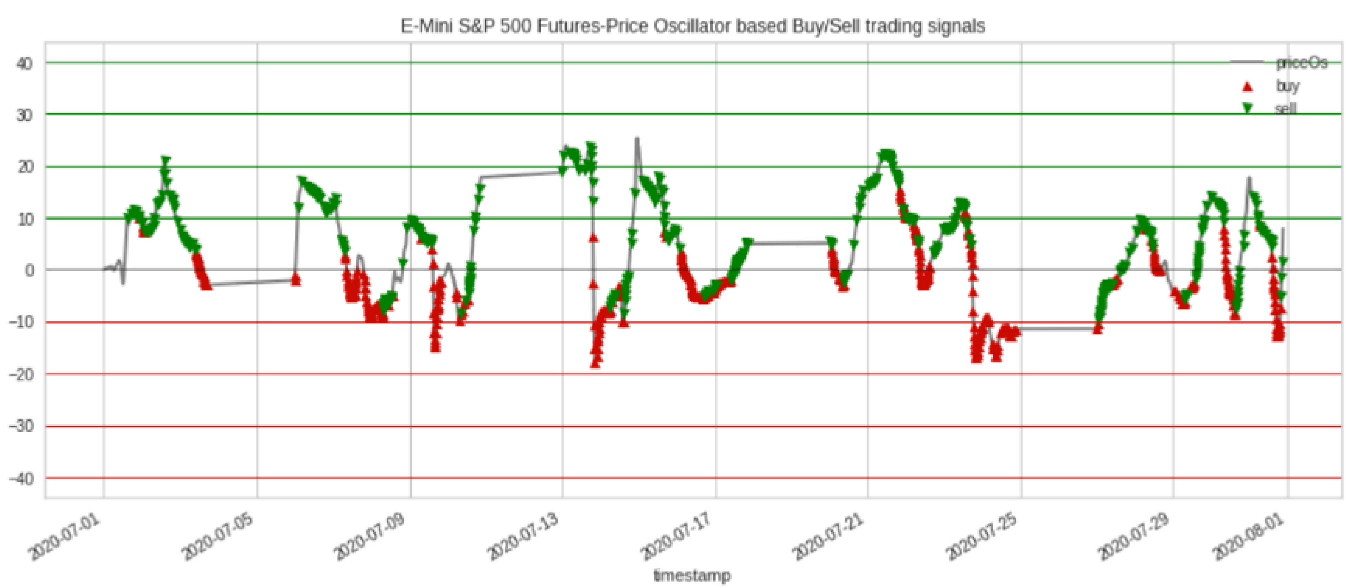
Preparing the data frame from the trading strategy results and visualizes the results:

```
data = df.copy()
data = data.assign(closePrice = pd.Series(close, index =df.index))
data = data.assign(short = pd.Series(fastEma_val, index = df.index))
data = data.assign(long = pd.Series(slowEma_val, index = df.index))
data = data.assign(priceOs = pd.Series(poValues, index = df.index))
data = data.assign(trades = pd.Series(orders, index = data.index))
data = data.assign(position = pd.Series(positions, index =
data.index))
data = data.assign(pnl = pd.Series(pnls, index=data.index))
data =
data[['closePrice','short','long','priceOs','trades','position','pnl'
]]
data.head(2)
```

	closePrice	short	long	priceOs	trades	position	pnl
timestamp							
2020-07-01 00:00:00	3081.169643	3081.169643	3081.169643	0.000000	0	0	0.0
2020-07-01 00:15:00	3081.633333	3081.170946	3081.169992	0.000953	0	0	0.0

```
plt.figure(figsize = (10, 6))
data['closePrice'].plot(color = 'gray',lw = 1., legend=True)
data['short'].plot(color ='green', lw = 1., legend=True)
data['long'].plot(color ='red', lw = 1., legend=True)
plt.title("E-Mini S &P 500 Futures - Market price")
plt.show()
```


```
plt.figure(figsize = (15, 6))
data['priceOs'].plot(color='gray', lw=2.,
legend=True)plt.plot(data.loc[data.trades == 1 ].index,
data.priceOs[data.trades == 1 ], color='r', lw=0, marker='^',
markersize=7, label='buy')
plt.plot(data.loc[data.trades == -1 ].index,
data.priceOs[data.trades == -1 ], color='green', lw=0, marker='v',
markersize=7, label='sell')
plt.axhline(y=0, lw=0.5, color='k')

for i in range(po_buyEnter, po_buyEnter * 5, po_buyEnter ):
  plt.axhline(y=i, lw=1., color='r')

for i in range(po_sellEnter, po_sellEnter * 5, po_sellEnter ):
  plt.axhline(y=i, lw=1., color='green')

plt.legend()
plt.title('E-Mini S&P 500 Futures-Price Oscillator based Buy/Sell
trading signals')
plt.show()
```



All signals are generated by the signal line crossing above zero-level which is considered bullish while crossing below zero-level is deemed bearish. As short-term momentum increases or decreases and eclipses long-term momentum the signal is generated. As is common with most oscillators, divergence in the price and indicator can also alert investors to early turnarounds.

```
plt.figure(figsize = (15, 6))
data['pnl'].plot(color='k', lw=1., legend=True)
plt.plot(data.loc[data.pnl > 1 ].index, data.pnl[data.pnl > 1 ],
color='green', lw=0, marker='.')
plt.plot(data.loc[data.pnl < 1 ].index, data.pnl[data.pnl < 1 ],
color='r', lw=0, marker='.')
plt.legend()
plt.title('P&Ls achieved by experimental strategy')
plt.show()
```

Key takeaways:

It is always advisable to experiment with other technical indicators prior to any official decision. As we have seen so far that, Price Oscillator is a momentum oscillator and measures the difference between two moving averages. Almost similar with MACD, the Price Oscillator can be shown with a signal line, a histogram and a centerline. Along with Price Oscillator, strategies e.g. candlestick patterns, triple EMA, volume indicator etc. are good to experiment. Volume Indicator is preferred by many because of the simple fact that, high volumes confirm well the entry signals generated by the price oscillator indicator, low volumes hint that the trend is being exhausted.

I can be reached [here](#).

Disclaimer: The programs described here are experimental and should not be used for any commercial purpose. All such use at your own risk.

Algorithmic Trading | Technical Indicator | Artificial Intelligence