Algorithmic Trading in Less than 100 Lines of Python Code

O'Reilly Media Follow
Dec 17, 2020 · 9 min read

If You Know Trading and Python, this Is Easy

Editor's Note: *The barriers to entry for algorithmic trading have never been lower. Not too long ago, only institutional investors with IT budgets in the millions of dollars could take part, but today even individuals equipped only with a notebook and an Internet connection can get started within minutes. In this piece from Python for Algorithmic Trading, author Yves Hilpisch shows you how to implement a complete algorithmic trading project, from backtesting the strategy to performing automated, real-time trading.*

Algorithmic trading refers to the computerized, automated trading of financial instruments (based on some algorithm or rule) with little or no human intervention during trading hours. Almost any kind of financial instrument — be it stocks, currencies, commodities, credit products or volatility — can be traded in such a fashion. Not only that, in certain market segments, algorithms are responsible for the lion's share of the trading volume. The books *The Quants* by Scott Patterson, *More Money Than God* by Sebastian Mallaby, and, more recently, *The Man Who Solved the Market* by Gregory Zuckerman paint a vivid picture of the beginnings of algorithmic trading and the personalities behind its rise.

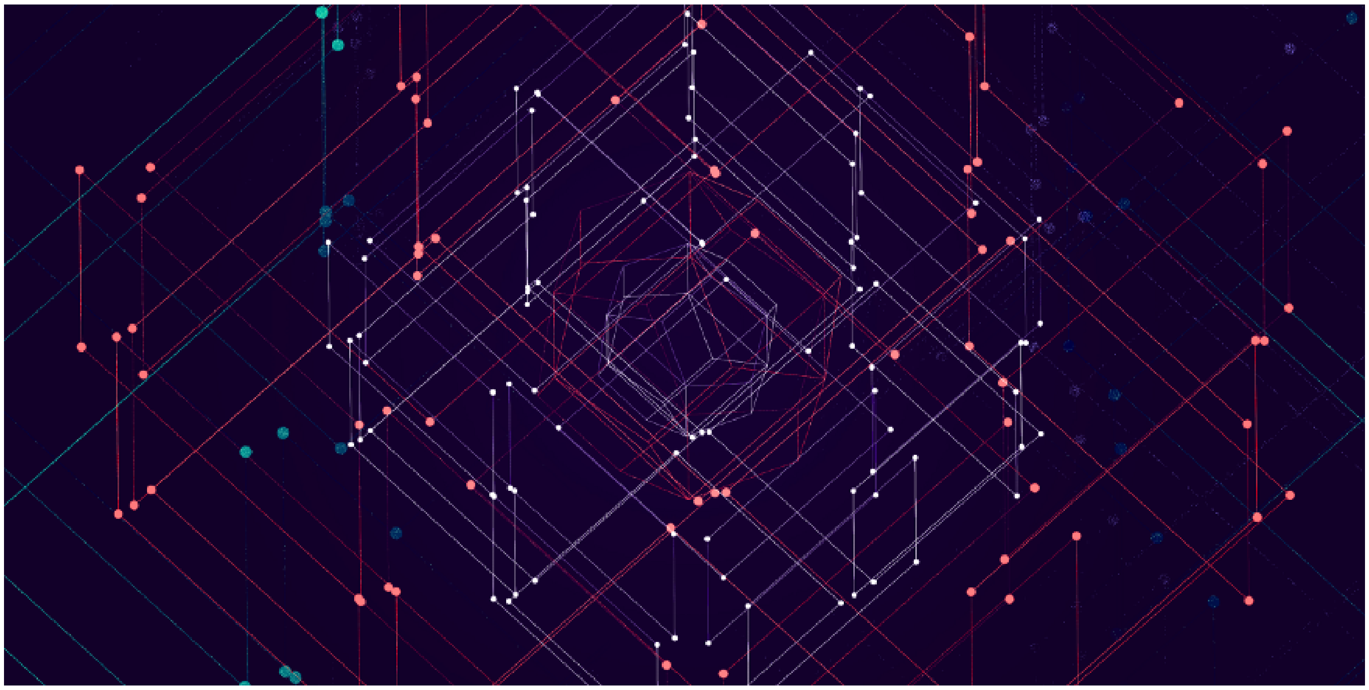
The barriers to entry for algorithmic trading have never been lower. Not too long ago, only institutional investors with IT budgets in the millions of dollars could take part, but today even individuals equipped only with a notebook and an Internet connection can get started within minutes. A few major trends are behind this development:

- **Open source software**: Every piece of software that a trader needs to get started in algorithmic trading is available in the form of open source; specifically, Python has become the language and ecosystem of choice.
- **Open data sources**: More and more valuable data sets are available from open and free sources, providing a wealth of options to test trading hypotheses and strategies.
- **Online trading platforms**: There are a large number of online trading platforms that provide easy, standardized access to historical data (via RESTful APIs) and real-time data (via socket streaming APIs), and also offer trading and portfolio features (via programmatic APIs).

This article shows you how to implement a complete algorithmic trading project, from backtesting the strategy to performing automated, realtime trading. Here are the major elements of the project:

- **Strategy**: I chose a time series momentum strategy (see Moskowitz, Tobias, Yao Hua Ooi, and Lasse Heje Pedersen (2012): "Time Series Momentum." Journal of Financial Economics, Vol. 104, 228–250.), which basically assumes that a financial instrument that has performed well/badly will continue to do so.
- **Platform**: I chose Oanda as the trading platform. It allows you to trade a variety of leveraged contracts for differences (CFDs, see also http://www.investopedia.com/terms/c/contractfordifferences.asp), which essentially allow for directional bets on a diverse set of financial instruments (such as currencies, stock indices, commodities).
- **Data**: We'll get all our historical data and streaming data from Oanda.
- **Software**: We'll use Python in combination with the powerful data analysis library `pandas` , plus a few additional Python packages.

The following assumes that you have a Python 3.6+ installation available with the major data analytics libraries, like NumPy and pandas, installed. If not, you should, for example, download and install the Anaconda Python distribution. Or even easier, sign up for free on the Quant Platform, execute your code in the cloud, and leverage lots of free Python content for algorithmic trading.



Oanda Account

At http://oanda.com, anyone can register for a free demo ("paper trading") account within minutes. Once you have done that, to access the Oanda API programmatically, you need to install the relevant Python package:

```
pip install git+https://github.com/yhilpisch/tpqoa
```

To work with the package, you need to create a configuration file with filename `oanda.cfg` in your current working directory that has the following content:

```
[oanda]
access_token = YOUR_ACCESS_TOKEN
account_id = YOUR_ACCOUNT_ID
account_type = practice
```

Replace the information above with the access token that you can generate and and the account ID that you find in your demo account on the Oanda platform.

The execution of the following code equips you with the main API object to work programmatically with the Oanda platform.

Backtesting

We have already set up everything n[illegible] with the backtesting of the momentum strategy. In particular, we are able to retrieve historical data from Oanda. The instrument we use is `EUR_USD` and is based on the EUR/USD exchange rate.

The first step in backtesting is to retrieve the data. The data set itself is for a single day, November 11, 2020, and has a granularity of one minute. Open (o), High (h), Low (l), Close © values — together OHCL — are returned for each minute which form the basis of a common one minute bar or candlestick type structure, along with Tick Volume (volume) and Complete (complete). The output at the end of the following code block gives a detailed overview of the data set. It is used to implement the backtesting of the trading strategy.

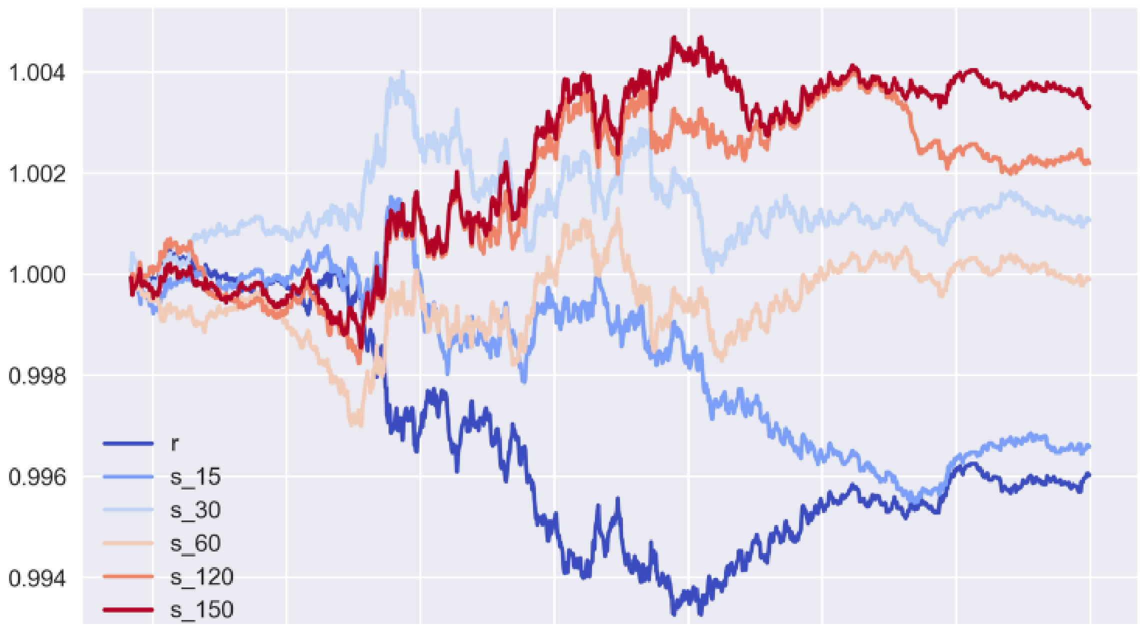
```
data = oanda.get_history(
    instrument='EUR_USD',
    start='2020-11-11',
    end='2020-11-12',
    granularity='M1',
    price='M'
)
data.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1424 entries, 2020-11-11 00:00:00 to 2020-11-11
23:59:00
Data columns (total 6 columns):

#       Column      Non-Null Count       Dtype
---     ------     --------------        -----
0         o        1424 non-null        float64
1         h        1424 non-null        float64
2         l        1424 non-null        float64
3         c        1424 non-null        float64
4       volume     1424 non-null        int64
5      complete    1424 non-null        bool
dtypes:   bool(1),   float64(4),   int64(1)
memory usage: 68.1 KB
```

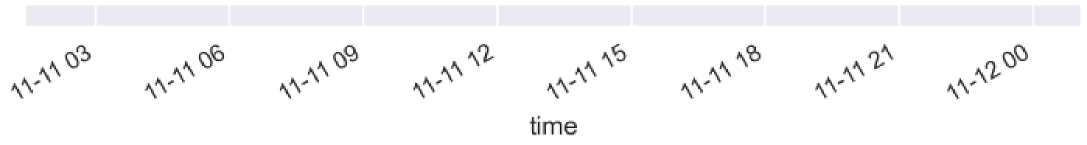
Second, we formalize the momentum strategy by telling Python to take the mean log return over the last 15, 30, 60, 120, and 150 minute bars to derive the position in the instrument. For example, the mean log return for the last 15 minute bars gives the average value of the last 15 return observations. If this value is positive, we go/stay long the traded instrument; if it is negative we go/stay short.

```
import numpy as np
data['r'] = np.log(data['c'] / data['c'].shift(1))
cols = []
for momentum in [15, 30, 60, 120, 150]:
    col = f'p_{momentum}'
    data[col] = np.sign(data['r'].rolling(momentum).mean())
    cols.append(col)
```

Third, to derive the gross performance of the momentum strategy for the different momentum intervals (in minutes), you need to multiply the positions (signals) derived above (shifed by one day) by the market returns. Here's how to do that:

```
from pylab import plt
plt.style.use('seaborn')
strats = ['r']
for col in cols:
    strat = f's_{col[2:]}'
    data[strat] = data[col].shift(1) * data['r']
    strats.append(strat)
data[strats].dropna().cumsum().apply(np.exp).plot(cmap='coolwarm');
```





Inspection of the plot above reveals that, over the period of the data set, the traded instrument itself has a negative performance. Among the momentum strategies, the one based on 150 minutes performs best with a positive return of about 0.35% (ignoring the bid/ask spread, see http://www.investopedia.com/ terms/b/bid-askspread.asp). In principle, this strategy shows "real alpha" (see also http://www.investopedia.com/terms/a/alpha.asp): it generates a positive return even when the instrument itself shows a negative one — while the risk as measured by the volatility stays the same for long/short strategies.

Automated Trading

Once you have decided on which trading strategy to implement, you are ready to automate the trading operation. To speed up things, I am implementing the automated trading based on twelve five-second bars for the time series momentum strategy instead of one-minute bars as used for backtesting. A single, rather concise class does the trick:

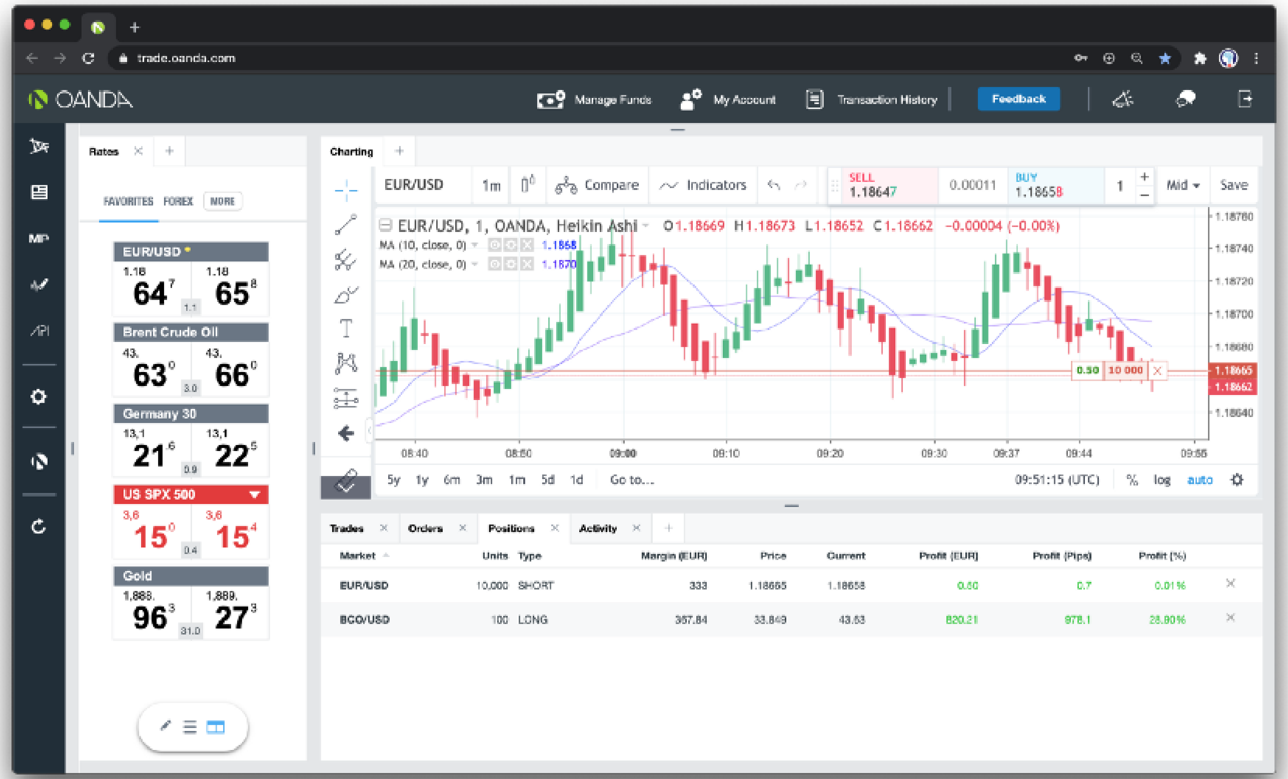
```
import pandas as pd
class MomentumTrader(tpqoa.tpqoa):
    def __init__(self, config_file, momentum):
        super(MomentumTrader, self).__init__(config_file)
        self.momentum = momentum
        self.min_length = momentum + 1
        self.position = 0  self.units = 10000
        self.tick_data = pd.DataFrame()
    def on_success(self, time, bid, ask):
        trade = False
        # print(self.ticks, end=' ')
        self.tick_data = self.tick_data.append(
            pd.DataFrame({'b': bid, 'a': ask, 'm': (ask + bid) / 2},
               index=[pd.Timestamp(time).tz_localize(tz=None)])
        )
        self.data = self.tick_data.resample('5s',
           label='right').last().ffill()
        self.data['r'] = np.log(self.data['m'] /
          self.data['m'].shift(1))
        self.data['m'] =
          self.data['r'].rolling(self.momentum).mean()
        self.data.dropna(inplace=True)
```

```
            if len(self.data) > self.min_length:
                self.min_length += 1
                if self.data['m'].iloc[-2] > 0 and self.position
                   in [0, -1]:
                  o = oanda.create_order(self.stream_instrument,
                                 units=(1 - self.position) * self.units,
                                 suppress=True, ret=True)
                  print('\n*** GOING LONG ***')
                  oanda.print_transactions(tid=int(o['id']) - 1)
                  self.position = 1
            if self.data['m'].iloc[-2] < 0 and self.position in [0, 1]:
                  o = oanda.create_order(self.stream_instrument,
                                units=-(1 + self.position) * self.units,
                                suppress=True, ret=True)
                  print('\n*** GOING SHORT ***')
                  self.print_transactions(tid=int(o['id']) - 1)
                  self.position = -1
```

The code below lets the `MomentumTrader` class do its work. The automated trading takes place on the momentum calculated over five intervals of length five seconds. The class automatically stops trading after 100 ticks of data received. This is arbitrary but allows for a quick demonstration of the `MomentumTrader` class.

```
mt = MomentumTrader('oanda.cfg', momentum=5)
mt.stream_data('EUR_USD', stop=100)

*** GOING SHORT ***
1975 | 2020-11-17T09:32:57.81 | EUR_USD | -10000.0 | 0.0

*** GOING LONG ***
1977 | 2020-11-17T09:33:15.83 | EUR_USD | 20000.0 | -2.2017

*** GOING SHORT ***
1979 | 2020-11-17T09:33:56.38 | EUR_USD | -20000.0 | -0.5928

*** GOING LONG ***
1981 | 2020-11-17T09:34:05.39 | EUR_USD | 20000.0 | -1.5241
```

The output above shows the single trades as executed by the `MomentumTrader` class during a demonstration run. During the execution of the trading algorithm, you can use Oandas trading tools to monitor the executed trades and open positions. The following screenshot shows the browser-based trade application — two positions are open.


trade.oanda.com
OANDA
Manage Funds
My Account
Transaction History
Feedback
Rates
FAVORITES FOREX MORE
EUR/USD 1.18 64 7 1.1 1.18 65 8
Brent Crude Oil 43. 63 0 3.0 43. 66 0
Germany 30 13,1 21 6 0.9 13,1 22 5
US SPX 500 3,6 15 0 0.4 3,6 15 4
Gold 1,888. 96 3 31.0 1,889. 27 3
Charting
EUR/USD 1m Compare Indicators
SELL 1.18647 0.00011 BUY 1.18658
Mid Save
EUR/USD, 1, OANDA, Heikin Ashi O1.18669 H1.18673 L1.18652 C1.18662 −0.00004 (−0.00%)
MA (10, close, 0) 1.1868
MA (20, close, 0) 1.1870
5y 1y 6m 3m 1m 5d 1d Go to...
09:51:15 (UTC) % log auto

Trades Orders Positions Activity

Market	Units	Type	Margin (EUR)	Price	Current	Profit (EUR)	Profit (Pips)	Profit (%)
EUR/USD	10,000	SHORT	333	1.18665	1.18658	0.60	0.7	0.01%
BCO/USD	100	LONG	367.84	33.849	43.63	820.21	978.1	28.90%

The following code closes out the final position and shows the complete, detailed order object.

```
from pprint import pprint
o = mt.create_order('EUR_USD', units=-mt.position * mt.units,
                    suppress=True, ret = True)
print('\n*** POSITION CLOSED ***')
mt.print_transactions(tid=int(o['id']) - 1)
print('\n')
pprint(o)
```

```
*** POSITION CLOSED ***
1987 | 2020-11-17T10:17:53.18 | EUR_USD | 10000.0 | -2.371

{'accountBalance': '98309.1911',
 'accountID': '101-004-13834683-001',
 'batchID': '1986',
 'commission': '0.0',
 'financing': '0.0',
 'fullPrice': {'asks': [{'liquidity': '10000000', 'price':
                   1.18693}],
              'bids': [{'liquidity': '10000000', 'price':
                   1.18679}],
              'closeoutAsk': 1.18693,
              'closeoutBid': 1.18679,
              'type': 'PRICE'},
'fullVWAP': 1.18693,
'gainQuoteHomeConversionFactor': '0.838346564431',
'guaranteedExecutionFee': '0.0',
'halfSpreadCost': '0.5898',
'id': '1987',
'instrument': 'EUR_USD',
'lossQuoteHomeConversionFactor': '0.846772158044',
'orderID': '1986',
'pl': '-2.371',
'price': 1.18693,
'reason': 'MARKET_ORDER',
'requestID': '78792003242162178',
'time': '2020-11-17T10:17:53.184866940Z',
'tradesClosed': [{'financing': '0.0',
                  'guaranteedExecutionFee': '0.0',
                  'halfSpreadCost': '0.5898',
                  'price': 1.18693,
                  'realizedPL': '-2.371',
                  'tradeID': '1985',
                  'units': '10000.0'}],
'type': 'ORDER_FILL',
'units': '10000.0',
'userID': 13834683}
```

All example outputs shown in this article are based on a demo account (where only paper money is used instead of real money) to *simulate* algorithmic trading. To move to a live trading operation with real money, you simply need to set up a real account with Oanda, provide real funds, and adjust the environment and account parameters used in the code. The code itself does not need to be changed.

However, you should be fully aware of the risks that such an endeavour entails and you should also have mastered the skills to manage such a transition to real trading on your own.

Conclusions

This article shows that you can start a basic algorithmic trading operation with fewer than 100 lines of Python code. In principle, all the steps of such a project are illustrated, like retrieving data for backtesting purposes, backtesting a momentum strategy, and automating the trading based on a momentum strategy specification. The code presented provides a starting point to explore many different directions: using alternative algorithmic trading strategies, trading alternative instruments, trading multiple instruments at once, and so forth. Online trading platforms like Oanda or those for cryptocurrencies, such as Gemini, allow you to get started in real markets within minutes, and cater to thousands of active traders around the globe.

Resources

To learn more about the skills required to apply Python for Algorithmic Trading you should consult these books and resources:

- Hilpisch, Yves. 2018. *Python for Finance: Mastering Data-Driven Finance*. 2nd ed. Sebastopol: O'Reilly.
- Hilpisch, Yves. 2020. *Artificial Intelligence in Finance: A Python-Based Guide*. Sebastopol: O'Reilly.
- Hilpisch, Yves. 2021. *Python for Algorithmic Trading: From Idea to Cloud Deployment*. Sebastopol: O'Reilly.
- Quant Platform with Jupyter Lab in the cloud — including free trial accounts and free Python code from the book Python for Algorithmic Trading
- Certificate Program in Python Algorithmic Trading by The Python Quants

Dr. Yves J. Hilpisch is founder and CEO of The Python Quants, a group focusing on the use of open source technologies for financial data science, artificial intelligence, algorithmic trading, and computational finance. He is also the founder and CEO of The AI Machine, a company focused on AI-powered algorithmic trading based on a proprietary strategy execution platform.

Yves has a Diploma in Business Administration (with distinction), a Ph.D. in Mathematical Finance (magna cum laude) and is Adjunct Professor for Computational Finance.

Yves is the author of five books:

- Python for Algorithmic Trading (2020, O'Reilly)
- Artificial Intelligence in Finance (2020, O'Reilly)
- Python for Finance (2018, 2nd ed., O'Reilly)
- Listed Volatility and Variance Derivatives (2017, Wiley Finance)
- Derivatives Analytics with Python (2015, Wiley Finance)

Yves is the director of the first online training program leading to University Certificates in Python for Algorithmic Trading and Computational Finance. He also lectures on computational finance, machine learning, and algorithmic trading at the CQF Program.

Yves is the originator of the financial analytics library DX Analytics and organizes Meetup group events, conferences, and bootcamps about Python, artificial intelligence, and algorithmic trading in London, New York, Frankfurt, Berlin, and Paris. He has given keynote speeches at technology conferences in the United States, Europe, and Asia.

Algorithms Algorithmic Trading