Hacking Google Finance in Real-Time for Algorithmic Traders

january 14, 2014 by [pawel](http://www.quantatrisk.com/author/admin/) [1 comment](http://www.quantatrisk.com/2014/01/14/hacking-google-finance-in-real-time-for-algorithmic-traders/#disqus_thread)

Forecasting risk in algorithmic stock trading is of paramount importance for everyone. You should always look for the ways how to detect sudden price changes and take immediate actions to protect your investments.

Imagine you opened a new long position last Wednesday for NASDAQ:NVDA buying 1500 shares at the market price of USD16.36. On the next day price goes down to USD15.75 at the end of the session. You are down 3.87% or almost a grand in one day. If you can handle that, it’s okay but if the drop were more steep? Another terrorist attack, unforeseen political event, North Korea nuclear strike? Then what? You need to react!

If you have information, you have options in your hands. In this post we will see how one can use real-time data of stock prices displayed on Google Finance website, fetch and record them on your computer. Having them, you can build your own warning system for sudden price swings (risk management) or run the code in the background for a whole trading session (for any stock, index, etc. accessible through Google Finance) and capture asset prices with an intraday sampling (e.g. every 10min, 30min, 1h, and so on). From this point only your imagination can stop you from using all collected data.

**Hacking with Python**

If you ever dreamt of becoming a hacker, this is your chance to shine! I have got my inspiration after reading the book of [Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers](http://www.amazon.com/Violent-Python-Cookbook-Penetration-Engineers/dp/1597499579/ref=sr_1_1?s=books&ie=UTF8&qid=1389556029&sr=1-1&keywords=violent+python) by TJ O’Connor. A powerful combination of the beauty and the beast.

The core of our code will be contained in a small function which does the job. For a specified Google-style ticker (query), it fetches the data directly from the server returning the most current price of an asset:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | *# Hacking Google Finance in Real-Time for Algorithmic Traders*  *#*  *# (c) 2014 QuantAtRisk.com, by Pawel Lachowicz*    **import** urllib, time, os, re, csv    **def** fetchGF(googleticker):  url="http://www.google.com/finance?&q="  txt=urllib.urlopen(url+googleticker).read()  k=re.search('id="ref\_(.\*?)">(.\*?)<',txt)  **if** k:  tmp=k.group(2)  q=tmp.replace(',','')  **else**:  q="Nothing found for: "+googleticker  **return** q |

Just make sure that a Google ticker is correctly specified (as will see below). Next, let’s display on the screen our local time and let’s force a change of the system time to the one corresponding to New York City, NY. The latter assumption we make as we would like to track the intraday prices of stock(s) traded at NYSE or NASDAQ. However, if you are tracking FTSE 100 index, the Universal Time (UTC) of London is advisable as an input parameter.

|  |  |
| --- | --- |
| 18  19  20  21  22  23  24  25  26  27 | *# display time corresponding to your location*  **print**(time.ctime())  **print**    *# Set local time zone to NYC*  os.environ['TZ']='America/New\_York'  time.tzset()  t=time.localtime() *# string*  **print**(time.ctime())  **print** |

Having that, let us define a side-function *combine* which we will use to glue all fetched data together into Python’s list variable:

|  |  |
| --- | --- |
| 29  30  31  32  33  34 | **def** combine(ticker):  quote=fetchGF(ticker) *# use the core-engine function*  t=time.localtime() *# grasp the moment of time*  output=[t.tm\_year,t.tm\_mon,t.tm\_mday,t.tm\_hour, *# build a list*  t.tm\_min,t.tm\_sec,ticker,quote]  **return** output |

As an input, we define Google ticker of our interest:

|  |  |
| --- | --- |
| 36 | ticker="NASDAQ:AAPL" |

for which we open a new text file where all queries will be saved in real-time:

|  |  |
| --- | --- |
| 39  40  41  42 | *# define file name of the output record*  fname="aapl.dat"  *# remove a file, if exist*  os.path.exists(fname) **and** os.remove(fname) |

Eventually, we construct the final loop over trading time. Here, we fetch the last data at 16:00:59 New York time. The key parameter in the game is *freq* variable where we specify the intraday sampling (in seconds). From my tests, using a private Internet provider, I have found that the most optimal sampling was 600 sec (10 min). Somehow, for shorter time intervals, Google Finance detected too frequent queries sent from my IP address. This test succeed from a different IP location, therefore, feel free to play with your local Internet network to find out what is the lowest available sampling time for your geolocation.

|  |  |
| --- | --- |
| 43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63 | freq=600 *# fetch data every 600 sec (10 min)*    **with** open(fname,'a') **as** f:  writer=csv.writer(f,dialect="excel") *#,delimiter=" ")*  **while**(t.tm\_hour<=16):  **if**(t.tm\_hour==16):  **while**(t.tm\_min<01):  data=combine(ticker)  **print**(data)  writer.writerow(data) *# save data in the file*  time.sleep(freq)  **else**:  **break**  **else**:  **for** ticker **in** tickers:  data=combine(ticker)  **print**(data)  writer.writerow(data) *# save data in the file*  time.sleep(freq)    f.close() |

To see how the above code works in practice, I conducted a test on Jan/9 2014, starting at 03:31:19 Sydney/Australia time, corresponding to 11:31:19 New York time. Setting the sampling frequency to 600 sec, I was able to fetch the data in the following form:

|  |
| --- |
| Thu Jan 9 03:31:19 2014    Wed Jan 8 11:31:19 2014    [2014, 1, 8, 11, 31, 19, '543.71']  [2014, 1, 8, 11, 41, 22, '543.66']  [2014, 1, 8, 11, 51, 22, '544.22']  [2014, 1, 8, 12, 1, 23, '544.80']  [2014, 1, 8, 12, 11, 24, '544.32']  [2014, 1, 8, 12, 21, 25, '544.86']  [2014, 1, 8, 12, 31, 27, '544.47']  [2014, 1, 8, 12, 41, 28, '543.76']  [2014, 1, 8, 12, 51, 29, '543.86']  [2014, 1, 8, 13, 1, 30, '544.00']  [2014, 1, 8, 13, 11, 31, 'Nothing found for: NASDAQ:AAPL']  [2014, 1, 8, 13, 21, 33, '543.32']  [2014, 1, 8, 13, 31, 34, '543.84']  [2014, 1, 8, 13, 41, 36, '544.26']  [2014, 1, 8, 13, 51, 37, '544.10']  [2014, 1, 8, 14, 1, 39, '544.30']  [2014, 1, 8, 14, 11, 40, '543.88']  [2014, 1, 8, 14, 21, 42, '544.29']  [2014, 1, 8, 14, 31, 45, '544.15']  ... |

As you can notice, they were displayed on the screen (line #59 in the code) in the form of Python’s list. It is important to note that the time we make an effort to capture and associate it with fetched asset price (query) is the computer’s system time, therefore please don’t expect regular time intervals as one may get from a verified market data providers. We are hacking in real-time! However, if you think about the data themselves, this time precision is not of great importance. As long as we fetch the data every *freq* seconds, that sufficiently allows us to build a risk management system or even to measure a rolling volatility of an asset. Your trading model will benefit anyway.

Have also a note that if our Internet connection fails or there are some disturbances of a different kind, we will miss the data in a sent query as visible in the example above.

Looks exciting? Give me *High Five!* and say *Hell Yeah!*

**Code Modification: Portfolio of Assets**

The presented Python code can be very easily modified if you wish to try fetching data for a couple of assets concurrently every *freq* seconds. Simply extend and amend all the lines starting at row #36, for example in the following form:

|  |  |
| --- | --- |
| 36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | tickers=["NASDAQ:AAPL","NASDAQ:GOOG","NASDAQ:BIDU","NYSE:IBM", \  "NASDAQ:INTC","NASDAQ:MSFT","NYSEARCA:SPY"]    *# define the name of an output file*  fname="portfolio.dat"  *# remove a file, if exist*  os.path.exists(fname) **and** os.remove(fname)    freq=600 *# fetch data every 600 sec (10 min)*    **with** open(fname,'a') **as** f:  writer=csv.writer(f,dialect="excel") *#,delimiter=" ")*  **while**(t.tm\_hour<=16):  **if**(t.tm\_hour==16):  **while**(t.tm\_min<01):  *#for ticker in tickers:*  data=combine(ticker)  **print**(data)  writer.writerow(data)  time.sleep(freq)  **else**:  **break**  **else**:  **for** ticker **in** tickers:  data=combine(ticker)  **print**(data)  writer.writerow(data)  time.sleep(freq)    f.close() |

That’s it! For the sake of real-time verification, here is a screenshot how does it work:

|  |
| --- |
| Thu Jan 9 07:01:43 2014    Wed Jan 8 15:01:43 2014    [2014, 1, 8, 15, 1, 44, 'NASDAQ:AAPL', '543.55']  [2014, 1, 8, 15, 1, 44, 'NASDAQ:GOOG', '1140.30']  [2014, 1, 8, 15, 1, 45, 'NASDAQ:BIDU', '182.65']  [2014, 1, 8, 15, 1, 45, 'NYSE:IBM', '187.97']  [2014, 1, 8, 15, 1, 46, 'NASDAQ:INTC', '25.40']  [2014, 1, 8, 15, 1, 47, 'NASDAQ:MSFT', '35.67']  [2014, 1, 8, 15, 1, 47, 'NYSEARCA:SPY', '183.43']  [2014, 1, 8, 15, 11, 48, 'NASDAQ:AAPL', '543.76']  [2014, 1, 8, 15, 11, 49, 'NASDAQ:GOOG', '1140.06']  [2014, 1, 8, 15, 11, 49, 'NASDAQ:BIDU', '182.63']  [2014, 1, 8, 15, 11, 50, 'NYSE:IBM', '187.95']  [2014, 1, 8, 15, 11, 51, 'NASDAQ:INTC', '25.34']  [2014, 1, 8, 15, 11, 52, 'NASDAQ:MSFT', '35.67']  [2014, 1, 8, 15, 11, 53, 'NYSEARCA:SPY', '183.34']  ... |

where we can see that we were able to grab the prices of 6 stocks and 1 ETF (Exchange Trading Fund tracking S&P500 Index) every 10 min.

**Reflection**

You may wonder whether hacking is legal or not? The best answer I find in the words of Gordon Gekko: *Someone reminded me I once said “Greed is good”*,

[](http://www.quantatrisk.com/wp-content/uploads/2014/01/gekko.jpg)

**PART II**  
    [Hacking Google Finance in Real-Time for Algo Traders. (2) Pre-Market Trading.](http://www.quantatrisk.com/2015/05/07/hacking-google-finance-in-pre-market-trading-python/)

filed under: [building a model](http://www.quantatrisk.com/category/building-model/), [data handling](http://www.quantatrisk.com/category/data-handling/), [python for quants](http://www.quantatrisk.com/category/python-for-quants/) tagged with:[hacking](http://www.quantatrisk.com/tag/hacking/), [python](http://www.quantatrisk.com/tag/python/)