Finance Research Letters 000 (2017) 1–9

Contents lists available at ScienceDirect

Finance Research Letters

journal homepage: www.elsevier.com/locate/frl

How to build a better database: When python programming meets Bloomberg's Open API

Adriano Durante, Eahab Elsaid[*]

Odette School of Business, University of Windsor, 401 Sunset Ave. Windsor, ON N9B3P4, Canada

A R T I C L E   I N F O

Article history:
Received 27 April 2017
Revised 20 June 2017
Accepted 5 July 2017
Available online xxx

JEL classifications:
G32
G34
M15

Keywords:
Bloomberg Open API
Python
Hand-collected data
SEC filings
Databases

A B S T R A C T

The need to hand collect data from SEC filings, among other sources, has long constituted a significant obstacle when conducting research in the area of finance (more specifically corporate finance) – and, indeed, business broadly defined. We propose a novel data collection alternative; using the Python programming language and Bloomberg's Open API we show how to automate the data collection process and generate databases of indefinite size. This is particularly useful when collecting data generally thought to be available only in proxy statements. Such variables include, but are not limited to: CEO age, CEO tenure, board size, number of independent and female directors on the board, and number of shares held by insiders. The approach described in our paper has significant implications for research, academia and in areas heretofore limited by the need to hand collect data.

1. Introduction

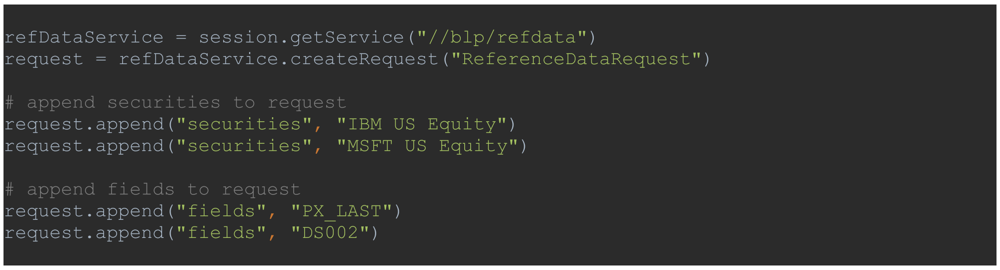
Time and money. Long have these constraints placed an upper limit on the amount of hand-collected data a researcher could reasonably hope to collect. A lack of one, the other, or both, renders our datasets smaller, our findings, in turn, less generalizable, and our insights ultimately less potent than they might otherwise have been. It is a situation made all the more intolerable when, as is so often the case, the data itself is readily available, if somewhat inconvenient to parse.

The typical SEC filing makes for an instructive case in point. As plentiful a source of information as any – particularly when it comes to so-called "environmental, social and governance" data – such documents, by their very nature, do not lend themselves to the timely extraction of the information contained within. It is a difficulty we have encountered in our own research involving executive succession and compensation, research for which the number of women who sit on the board of directors is a key variable of interest. To obtain this figure, our customary workflow was as follows: 1) *acquire* the relevant proxy statement(s) for the firms in question; 2) *read* the proxy statement (or at least its Election of Directors section); 3) *count* the number of women present, and; 4) *record* the sum in a spreadsheet.

A nettlesome and time-consuming process to be sure, it is also one that can well prove prohibitive. Our time – and that of our costly research assistants – being limited, it is not difficult to imagine cases for which it is simply not feasible to collect the requisite data, at least not in a volume sufficient to allow for the drawing of statistically meaningful conclusions.

* Corresponding author.
E-mail addresses: durantea@uwindsor.ca (A. Durante), elsaid@uwindsor.ca (E. Elsaid).

http://dx.doi.org/10.1016/j.frl.2017.07.006
1544-6123/© 2017 Elsevier Inc. All rights reserved.



```
refDataService = session.getService("//blp/refdata")
request = refDataService.createRequest("ReferenceDataRequest")

# append securities to request
request.append("securities", "IBM US Equity")
request.append("securities", "MSFT US Equity")

# append fields to request
request.append("fields", "PX_LAST")
request.append("fields", "DS002")
```

Fig. 1. Code snippet showing the creation of a reference data request.

This is why datasets featuring hand-collected data are so valuable; given the difficultly with which these datasets are assembled, whatever insights may be gleaned from them are likewise difficult to replicate. And a unique insight is, of course, a publishable insight.

That there exists some set of questions off-limits to all but our most well-funded colleagues is a basic, if perhaps regrettable, fact of academia, the need to hand collect data being too significant a barrier to entry for the rest of us. Hence the significance of what we propose here.

Using the Bloomberg Open API and the Python programming language (Python 2.7 to be exact), we show that a great many variables generally thought to be available only in SEC filings need not be hand collected at all. These include – but are by no means limited to – board size, annual number of board meetings, average meeting attendance, the number of women and independent directors on the board, average age and tenure of directors, as well as ownership and compensation details for these and other insiders. Such is but a small fraction of the data available through the Bloomberg Professional Service, more commonly known as a Bloomberg Terminal. While Bloomberg's Open API allows us to interact with the Terminal *programmatically*, we use Python to determine the nature of that interaction. In this way, we are able to automate the whole of the data collection process.

In what follows, we describe the mechanics of our program in some technical detail. We also discuss some interesting use cases for which the tools we have developed can be easily adapted. Perhaps needless to say, these tools can be used to gather all manner of financial measures and statistics, not simply those relating to corporate governance; but for reasons given above, we will frame our discussion in terms of those variables that are not easily acquired elsewhere, those that are generally thought to require hand collection.

2. An overview of Bloomberg's Open API

While the capabilities of Bloomberg's API are many – indeed, the vast majority of which are beyond the scope of this paper – for our purposes, we need only consider the API's "reference data service." Using a request/response paradigm, this service gives us access to what Bloomberg calls "Reference Data" – i.e., the current value of whatever data point we happen to request – as well as "Historical End-of-Day Data" – i.e., end-of-day values for the data in question over a given period of time (Bloomberg, 2016).

Bloomberg helpfully provides starter code to illustrate this process. Although it will only run on a Bloomberg Terminal, the code itself is freely available at the website https://www.bloomberg.com/professional/support/api-library/. Simply download and unzip the "Python Supported Release" zip file, and there we find an "examples" folder full of example code. (We also find the "blpapi" folder which gives us ready access to the API's source code, a resource well worth consulting.)

Let us consider the following code snippet from the *SimpleRefDataExample.py* file found in the example folder mentioned above (Fig. 1):

In the first two lines we can see the creation of a "ReferenceDataRequest." In the next two lines we can see the securities we are investigating; in this case, IBM and Microsoft. In the last two lines we can see what it is we want to know about these companies. This is done by specifying what "field" we are interested in. A field can be referenced either by its mnemonic – "PX_LAST" being the last price – or its field ID – "DS002" being the name of the company.

Analogous code for a "HistoricalDataRequest," found in the *SimpleHistoryExample.py* file, can be seen in the following code snippet (Fig. 2):

The principle difference when making a "HistoricalDataRequest" is the need to set additional parameters, a start and end date being primary examples. Notice however that in either case, the securities under investigation have been hard-coded into the program. As written, each security occupies its own line of code; to proceed in this manner would be most inefficient, especially if we wish to investigate a large number of securities.

Nor is the response to our data request particularly helpful in its default format. Delivered in one or more response "messages," these take the following form (Fig. 3):

What we have here is a nested data structure. Notice that the values of interest to us appear as elements of the "fieldData" container. The "fieldData" container is however itself an element of a higher level container, which is itself an element of yet a higher level container still. In order to make use of these values, they must first be extracted from the response mes-