IRS 990 e-File Data (8) – Data Wrangling and Export to Stata

IRS 990 e-File Data (7) – Generate Codebook

IRS 990 e-File Data (6) – Download IRS 990, 990EZ, and 990PF Filings and Associated Schedules into MongoDB

IRS 990 e-File Data (part 5)

– Download IRS 990 Filings and Associated Schedules into MongoDB

IRS 990 e-File Data (part 4) – Download IRS 990 Filings into MongoDB

IRS 990 e-File Data (part 3) – Load Index Files into PANDAS

IRS 990 e-File Data (part 2)

– Load Index Data and Insert into MongoDB

Tutorials for Sarbanes-Oxley Paper Data

Dan Neely (from University of Milwaukee-Wisconsin) and I just had the following article published at the *Journal of Business Ethics*:

Saxton, G. D., & Neely, D. G. (2018). The Relationship Between Sarbanes–Oxley Policies and Donor Advisories in Nonprofit Organizations. *Journal of Business Ethics.*

This page contains tutorials on how to download the IRS 990 e-file data that was used for the control variables in our study.

Tutorials

- IRS 990 e-File Data (part 1) – Set up AWS CLI credentials and grab index files
- IRS 990 e-File Data (part 2) – Load Index Data and Insert into MongoDB
- IRS 990 e-File Data (part 3) – Load Index Files into PANDAS
- IRS 990 e-File Data (part 4) – Download IRS 990 Filings into MongoDB
- IRS 990 e-File Data (part 5) – Download IRS 990 Filings

and Schedules into MongoDB
- IRS 990 e-File Data (part 6) – Download IRS 990, 990PF, and 990EZ Filings into MongoDB
- IRS 990 e-File Data (part 7) – Generate Data Codebook
- IRS 990 e-File Data (part 8) – Data Wrangling and Export to Stata

I hope you have found this helpful. If so, please spread the word, and happy coding!

Downloading Tweets, Take III – MongoDB

In this tutorial I walk you through how to use Python and MongoDB to download tweets from a list of Twitter users.

This tutorial builds on several recents posts on how to use Python to download Twitter data. Specifically, in a previous post I showed you how to download tweets using Python and an SQLite database – a type of traditional *relational* database. More and more people are interested in *noSQL* databases such as MongoDB, so in a follow-up post I talked about the advantages and disadvantages of using SQLite vs MongoDB to download social media data for research purposes. Today I go into detail about how to actually use MongoDB to download your data and I point out the differences from the SQLite approach along the way.

Overview

This tutorial is directed at those who are new to Python, MongoDB, and/or downloading data from the Twitter API. We will be using Python to download the tweets and will be inserting

the tweets into a MongoDB database. This code will allow you to download up to the latest 3,200 tweets sent by each Twitter user. I will not go over the script line-by-line but will instead attempt to provide you a 'high-level' understanding of what we are doing – just enough so that you can run the script successfully yourself.

Before running this script, you will need to:

- Have Anaconda Python 2.7 installed
- Have your Twitter API details handy
- Have MongoDB installed and running
- Have created a CSV file (e.g., in Excel) containing the Twitter handles you wish to download. Below is a sample you can download and use for this tutorial. Name it *accounts.csv* and place it in the same directory as the Python script.

If you are completely new to Python and the Twitter API, you should first make your way through the following tutorials, which will help you get set up and working with Python:

- Which version of Python to download
- Running your first code
- Four ways to run your code
- Setting up access to the Twitter API

Another detailed tutorial I have created, Python Code Tutorial, is intended to serve as an introduction to how to access the Twitter API and then read the JSON data that is returned. It will be helpful for understanding what we're doing in this script.

Also, if you are not sure you want to use MongoDB as your database, take a look at this post, which covers the advantages and disadvantages of using SQLite vs MongoDB to download social media data. As noted in that post, MongoDB has a more detailed installation process.

At the end of this post I'll show the entire script. For now, I'll go over it in sections. The code is divided into seven parts:

Part I: Importing Necessary Python Packages

The first line in the code is the shebang – you'll find this in all Python code.

Lines 3 – 23 contain the docstring – also a Python convention. This is a multi-line comment that describes the code. For single-line comments, use the # symbol at the start of the line.

In lines 26 – 31 we'll import some Python packages needed to run the code. Twython can be installed by opening your Terminal and installing by entering pip install Twython. For more details on this process see this blog post.

Part II: Import Twython and Twitter App Key and Access Token

Lines 37-42 is where you will enter your Twitter *App Key* and *Access Token* (lines 40-41). If you have yet to do this you can refer to the tutorial on Setting up access to the Twitter API.

Part III: Define a Function for Getting Twitter Data

In this block of code we are creating a Python *function*. The

function sets up which part of the Twitter API we wish to access (specifically, it is the get user timeline API), the number of tweets we want to get per page (I have chosen the maximum of 200), and whether we want to include retweets. We will call this function later on in the code.

Part IV: Set up MongoDB Database and Collections (Tables)

Lines 72-111 are where you set up your MongoDB database and 'collections' (tables).

This is where you'll see the first major differences from an SQLite implementation of this code. First, unlike SQLite, you will need to make sure MongoDB is *running* by typing mongod or sudo mongod in the terminal. So, that's one extra step you have to take with MongoDB. If you're running the code on a machine that is running 24/7 that is no issue; if not you'll just have to remember.

There is a big benefit to MongoDB here, however. Unlike with the SQLite implementation, there is no need to pre-define every column in our database tables. As you can see in the SQLite version, we devoted 170 lines of code to defining and naming database columns.

Below, in contrast, we are simply making a connection to MongoDB, creating our database, then our database tables, then indexes on those tables. Note that, if this is the first time you're running this code, the database and tables and indexes will be *created*; if not, the code will simply access the database and tables. Note also that MongoDB refers to database tables as 'collections' and refers to columns or variables as 'fields.'

One thing that is similar to the SQLite version is that we are setting indexes on our database tables. This means that no two

tweets with the same index value – the tweet's ID string (*id_str*) – can be inserted into our database. This is to avoid duplicate entries.

One last point: we are setting up two tables, one for the tweets and one to hold the Twitter account names for which we wish to download tweets.

Part V: Read in Twitter Accounts (and add to MongoDB database if first run)

In Lines 117-139 we are creating a Python *list* of Twitter handles for which we want to download tweets. The first part of the code (lines 119-130) is to check if this is the first time you're running the code. If so, it will read the Twitter handle data from your local CSV file and insert it into the *accounts* table in your MongoDB database. In all subsequent runs of the code the script will skip over this block and go directly to line 137 – that creates a list called *twitter_accounts* that we'll loop over in Part VI of the code.

Part VI: Main Loop: Loop Over Each of the Twitter Handles in the *Accounts* Table and Download Tweets

In lines 144-244 we are at the last important step.

This code is much shorter here as well compared to the SQLite version. As noted in my previous post comparing SQLite to MongoDB, in MongoDB we do not need to define all of the columns we wish to insert into our database. MongoDB will just take whatever columns you throw at it and insert. In the SQLite version, in contrast, we had to devote 290 lines of code just specifying what specific parts of the Twitter data we are grabbing and how they relate to our pre-defined variable names.

After stripping out all of those details, the core of this code is the same as in the SQLite version. At line 151 we begin a *for loop* where we are looping over each Twitter ID (as indicated by the *Twitter_handle* variable in our *accounts* database).

Note that within this *for loop* we have a *while loop* (lines 166-238). What we are doing here is, for each Twitter ID, we are grabbing up to 16 pages' worth of tweets; this is the maximum allowed for by the Twitter API. It is in this loop (line 170) that we call our *get_data_user_timeline_all_pages* function, which on the first loop will grab page 1 for the Twitter ID, then page 2, then page 3, …. up to page 16 so long as there are data to return.

Lines 186-205 contains code for writing the data into our MongoDB database table. We have defined our variable *d* to contain the result of calling our *get_data_user_timeline_all_pages* function – this means that, if successful, *d* will contain 200 tweets' worth of data. The *for loop* starting on line 187 will loop over each tweet, add three variables to each tweet – *date_inserted*, *time_date_inserted*, and *screen_name* – and then insert the tweet into our *tweets* collection.

One last thing I'd like to point out here is the API limit checks I've written in lines 221-238. What this code is doing is checking how many remaining API calls you have. If it is too low, the code will pause for 5 minutes.

Part VII: Print out Number of Tweets in Database per Account

This final block of code will print out a summary of how many tweets there are per account in your *tweets* database.

Now let's put the whole thing together. To recap, what this

entire script does is to loop over each of the Twitter accounts in the *accounts* table of your MongoDB database – and for each one it will grab up to 3,200 tweets and insert the tweets into the *tweets* table of your database.

Below is the entire script – download it and save it as *tweets.py* (or something similar) in the same directory as your *accounts.csv* file. Add in your Twitter API account details and you'll be good to go! For a refresher on the different ways you can run the script see this earlier post.

If you've found this post helpful please share on your favorite social media site.

You're on your way to downloading your own Twitter data! Happy coding!