Machine Reading IMF Data: Data Retrieval with Python

Brian Dew, brianwdew@gmail.com

March 22, 2016

1 Introduction

The International Monetary Fund (IMF) Statistics Department (STA) allows API access to their economic time series. Well-known datasets such as International Financial Statistics (IFS) can be machine read through the API. This example will use Python to retrieve Direction of Trade Statistics (DOTS) data from STA's JSON RESTful Web Service so that we can determine the United States' share of world exports over the past 50 years.

The IMF knowledge base provides more information on the three available API formats and IMF data services. For more information on the work of STA, see their annual report (PDF), STA at a glance 2015.

2 Gathering series and dimension information

First, we will need to import the json, urllib2, and requests libraries, as well as pandas and numpy. These will allow us to read json data, open urls, and request information from the web.

```
In [1]: # Import libraries
        import json
        import urllib2
        import requests
        import pandas as pd
        import numpy as np
```

Since we are using the JSON RESTful API, we start by using the 'Dataflow' endpoint URL to look at what series are available and find the series id of interest. The full output is long, so I've removed the data unrelated to this example. The IMF has many more series than what is shown below.

```
In [2]: # Find the series id and text name.
        url = "http://dataservices.imf.org/REST/SDMX_JSON.svc/Dataflow/"
        seriesids = json.load(urllib2.urlopen(url))
        df = pd.DataFrame(seriesids['Structure']['KeyFamilies']['KeyFamily'])
        for x in range(6, 13):
            items = (str(df['@id'][x]), str(df['Name'][x]['#text']))
            print ': '.join(items)
```

```
FSI: Financial Soundness Indicators (FSI)
DOT: Direction of Trade Statistics (DOTS)
FSIREM: Financial Soundness Indicators (FSI), Reporting Entities - Multidimensional
CDIS: Coordinated Direct Investment Survey (CDIS)
GFS01M: Government Finance Statistics (GFS 2001) - Multidimensional
GFS01: Government Finance Statistics (GFS 2001)
BOP: Balance of Payments (BOP)
```

We found above that the id for Direction of Trade Statistics is DOT. We can use this id to read notes about the series. We will next need to identify the *dimensions* of the data. For example, direction of trade data is based on a home country, a flow of goods, and a counterpart country. DOTS data is also available with multiple frequencies and units of measurement. All of this dimensional information will be needed later to make our data request.

```
In [3]: # Annotations for the series
        url = "http://dataservices.imf.org/REST/SDMX_JSON.svc/DataStructure/DOT"
        dotstruct = json.load(urllib2.urlopen(url))
        df = pd.DataFrame(dotstruct['Structure']['KeyFamilies']\
                          ['KeyFamily']['Annotations'])
        for x in range(0, 7):
            items = (str(df['Annotation'][x]['AnnotationTitle']), \
                     str(df['Annotation'][x]['AnnotationText']['#text']))
            print ': '.join(items)
```

```
Latest Update Date: 02/24/2016
Name: Direction of Trade Statistics (DOTS)
Temporal Coverage: Monthly and quarterly data are available starting 1960. Annual data
are available starting 1947.
Geographic Coverage: DOTS covers 184 countries, the world, and major areas.
Methodology: Guide to Direction of Trade Statistics, 1993. See Documents tab.
Definition: The <B>Direction of Trade Statistics (DOTS)</B> presents current figures on
the value of merchandise exports and imports disaggregated according to a country's primary
trading partners. Area and world aggregates are included in the display of trade flows
between major areas of the world. Reported data is supplemented by estimates whenever such
data is not available or current. Imports are reported on a cost, insurance and freight
(CIF) basis and exports are reported on a free on board (FOB) basis, with the exception
of a few countries for which imports are also available FOB. Time series data includes
estimates derived from reports of partner countries for non-reporting and slow-reporting
countries.
Code: DOT
```

```
In [4]: # Look at structure of DOTS data to find the dimensions for our data request
        url = "http://dataservices.imf.org/REST/SDMX_JSON.svc/DataStructure/DOT"
        dotstruct = json.load(urllib2.urlopen(url))
        df = pd.DataFrame(dotstruct['Structure']['KeyFamilies']['KeyFamily']\
                          ['Components']['Dimension'])
        for x in range(0, 5):
            items = ("Dimension", str(x+1), str(df['@codelist'][x]))
            print ': '.join(items)
```

```
Dimension: 1: CL_COUNTRY|DOT
Dimension: 2: CL_INDICATOR|DOT
Dimension: 3: CL_COUNTERPART_COUNTRY|DOT
Dimension: 4: CL_FREQ|DOT
Dimension: 5: CL_UNIT_MULT|DOT
```

We can now copy the code for each dimension into the CodeList Method to get the list of possible values. For example, we will need to identify the value of the first dimension, CL_COUNTRY, for the United States. Below, we show that the code is 111. I've manually placed the index number for the U.S. and World codes (again to save space), however, you can replace [200, 247] with range(0, 247) to get the full list of country/area codes.

```
In [5]: # Obtain country codes
        url = "http://dataservices.imf.org/REST/SDMX_JSON.svc/CodeList/CL_COUNTRY|DOT"
```

```
        country = json.load(urllib2.urlopen(url))
        df = pd.DataFrame(country['Structure']['CodeLists']['CodeList']['Code'])
        for x in [200, 247]:
            items = (str(df['@value'][x]), str(df['Description'][x]['#text']))
            print ': '.join(items)
```

```
111: United States
001: World
```

The series ID is DOT and the country codes (we will use this with the exporting country, CL_COUNTRY, and the counterpart, CL_COUNTERPART_COUNTRY) of interest are 001 for world and 111 for the US. We see below that the indicator of interest is TXG_FOB_USD, Goods, Value of Exports, Free on board (FOB), US Dollars.

```
In [6]: # Obtain series info and ids
        url = "http://dataservices.imf.org/REST/SDMX_JSON.svc/CodeList/CL_INDICATOR|DOT"
        series = json.load(urllib2.urlopen(url))
        df = pd.DataFrame(series['Structure']['CodeLists']['CodeList']['Code'])
        for x in range(0, 4):
            items = (str(df['@value'][x]), str(df['Description'][x]['#text']))
            print ': '.join(items)
```

```
TXG_FOB_USD: Goods, Value of Exports, Free on board (FOB), US Dollars
TMG_CIF_USD: Goods, Value of Imports, Cost, Insurance, Freight (CIF), US Dollars
TMG_FOB_USD: Goods, Value of Imports, Free on board (FOB), US Dollars
All Indicators: All Indicators
```

We repeat the above steps for each dimension and record which series values are of interest to us.

3 Retrieving Data

The guide to STA's API shows how we can combine information from the previous steps to call and retrieve data. For direction of trade statistics, we see that the dimensions are as follows:

- Dimension 1: REF_AREA (the primary country) - 111
- Dimension 2: INDICATOR (the measure–we want to look at exports free of board) - TXG_FOB_USD
- Dimension 3: VIS_AREA (the counterpart country) - 001
- Dimension 4: FREQ (the frequency of the data–we want to use monthly data) - M
- Dimension 5: SCALE (the units of measure–we can leave this blank)

The first dimension is the exporting country, in this example. We are interested initially in the United States, which has country code 111. The second dimension is the indicator. As discussed, we will look at exports free of board. The third dimension is the importer country. We are interested in exports from the U.S. to all countries in the rest of the world, so we use the code 001. The fourth dimension is the frequency of the data, and we are interested in monthly data, which is called with the letter M. Lastly, the fifth dimension covers the scale, which we can leave blank, as the indicator from dimension two has already identified the scale as US Dollars.

The JSON RESTful API method for requesting the data is the CompactData Method. The format for putting together dimension and time period information is shown on the Web Service knowledge base as:

```
http://dataservices.imf.org/REST/SDMX_JSON.svc/CompactData/{database ID}/ {item1
from dimension1}+{item2 from dimension1}{item N from dimension1}.{item1 from
dimension2} +{item2 from dimension2}+{item M from dimension2}? startPeriod={start
date}&endPeriod={end date}
```

Putting all of this information together, the URL to retrieve a JSON dictionary for 1966-2016 US exports to the world data is:

http://dataservices.imf.org/REST/SDMX_JSON.svc/CompactData/DOT/
111.TXG_FOB_USD.001.M.?startPeriod=1966&endPeriod=2016

The Python code which gets the data and saves it as a dictionary is as follows:

```
In [7]: url = 'http://dataservices.imf.org/REST/SDMX_JSON.svc/CompactData/DOT/111...'
        data = json.loads(requests.get(url).text)
        usexp = pd.DataFrame(data['CompactData']['DataSet']['Series']['Obs'])
        usexp.columns = ['date','usexports'];
        usexp.tail()
```

```
Out[7]:         date     usexports
        593  2015-06  131003226848
        594  2015-07  124169981027
        595  2015-08  123065722863
        596  2015-09  125394021087
        597  2015-10  130599510034
```

It is critical in the above example to use the full URL. The listed URL has been truncated to allow the code to fit on one line.

We can repeat the above code with a different URL to obtain data on total world exports and the exports of other countries which we may want to compare with the United States. We combine the request for several series into one URL, by adding '+code2+code3'. For example, '001+998.TXG..'

The full URL for our second request (again, I've shorted the url in the code to fit on one line) is as follows:

http://dataservices.imf.org/REST/SDMX_JSON.svc/CompactData/DOT/
001+998+924+158.TXG_FOB_USD.001.M.?startPeriod=1966&endPeriod=2016

```
In [8]: ourl = 'http://dataservices.imf.org/REST/SDMX_JSON.svc/CompactData/DOT/001+998...'
        odata = json.loads(requests.get(ourl).text);
```

```
In [9]: eexp = pd.DataFrame(odata['CompactData']['DataSet']['Series'][0]['Obs'])
        eexp.columns = ['observ','date','euroexports']
        del eexp['date']
        del eexp['observ']
        wexp = pd.DataFrame(odata['CompactData']['DataSet']['Series'][1]['Obs'])
        wexp.columns = ['observ','date','worldexports']
        del wexp['date']
        del wexp['observ']
        cexp = pd.DataFrame(odata['CompactData']['DataSet']['Series'][2]['Obs'])
        cexp.columns = ['observ','date','chinaexports']
        del cexp['date']
        del cexp['observ']
        jexp = pd.DataFrame(odata['CompactData']['DataSet']['Series'][3]['Obs'])
        jexp.columns = ['observ','date','japanexports']
        del jexp['date']
        del jexp['observ'];
```

Now we combine the two series into one pandas dataframe and tell our script to read the export value columns as numbers.

```
In [10]: combined = pd.concat([usexp, wexp, eexp, cexp, jexp], axis=1)
         pd.to_datetime(combined.date)
         combined = combined.set_index(pd.DatetimeIndex(combined['date']))
         usexports = pd.to_numeric(combined.usexports)
         wexports = pd.to_numeric(combined.worldexports)
         eexports = pd.to_numeric(combined.euroexports)
         cexports = pd.to_numeric(combined.chinaexports)
         jexports = pd.to_numeric(combined.japanexports)
```

Finally, we can calculate the U.S. percentage share of world exports. We simply divide the us exports by the world exports and multiply by 100. If using the data for economic research, we would likely take the log forms and apply some filters.

$$\frac{\text{EX}_{US}}{\text{EX}_{world}} * 100$$

```
In [11]: combined['usshare'] = usexports / wexports * 100
         combined['euroshare'] = eexports / wexports * 100
         combined['chinashare'] = cexports / wexports * 100
         combined['japanshare'] = jexports / wexports * 100
         combined.tail()
```

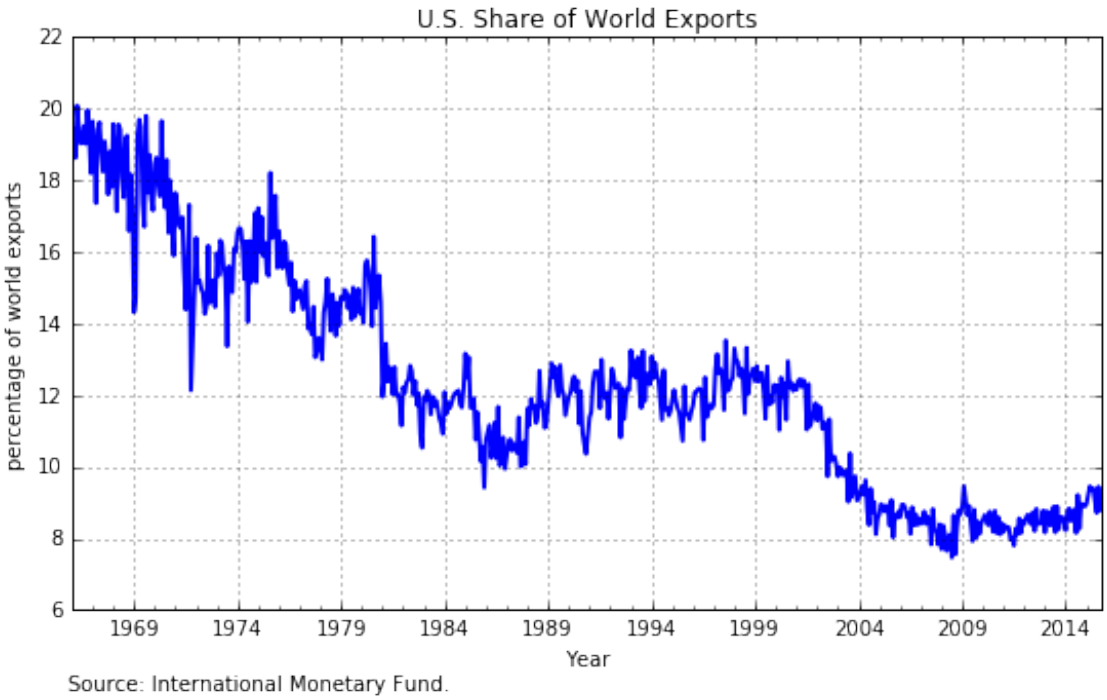
```
Out[11]:                date     usexports      worldexports   euroexports  \
         2015-06-01  2015-06  131003226848  1450418752716.69  485806768270
         2015-07-01  2015-07  124169981027  1424179666172.93  467762777592
         2015-08-01  2015-08  123065722863  1302351524099.28  385488153410
         2015-09-01  2015-09  125394021087  1425978957082.99  471978522421
         2015-10-01  2015-10  130599510034   1485158888055.6  480836047976

                     chinaexports      japanexports   usshare  euroshare  \
         2015-06-01      191791436215  52570900704.1295  9.032097  33.494242
         2015-07-01      194982859280  54069054169.2588  8.718702  32.844366
         2015-08-01      196700577701   47728520490.666  9.449501  29.599394
         2015-09-01      205276102549  53348138188.4691  8.793539  33.098562
         2015-10-01  224042809635.292  59220087798.3025  8.793639  32.376068

                     chinashare  japanshare
         2015-06-01   13.223177    3.624533
         2015-07-01   13.690889    3.796505
         2015-08-01   15.103493    3.664796
         2015-09-01   14.395451    3.741159
         2015-10-01   15.085444    3.987458
```

4 Graphing the data

Let's use matplotlib to view the result of our work.

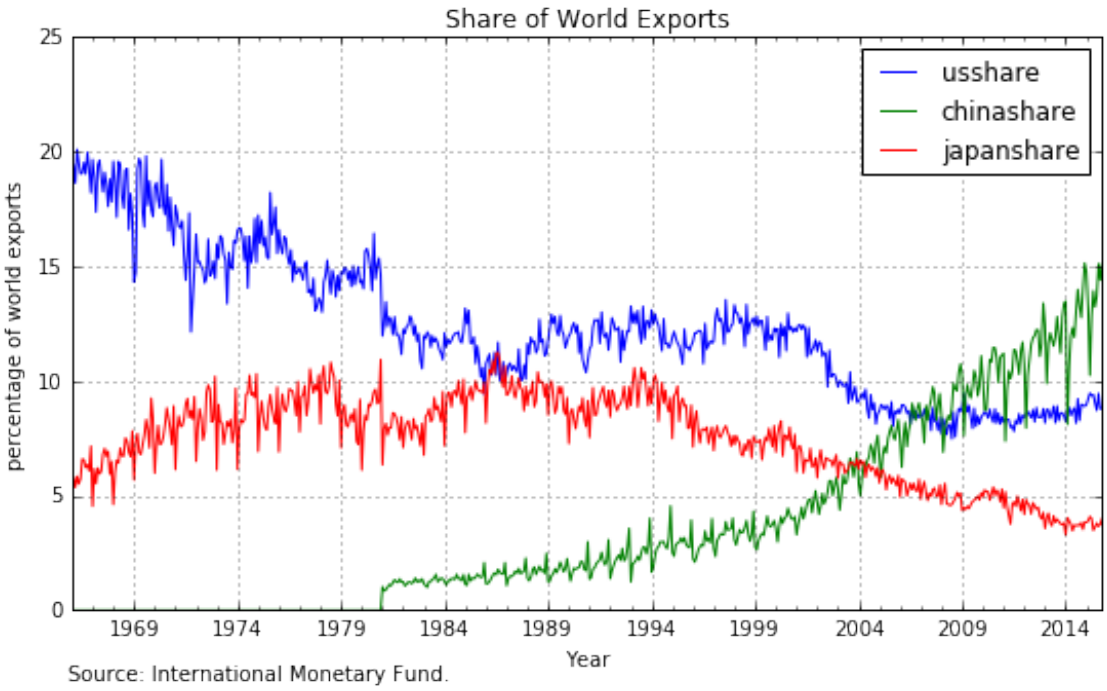
```
In [12]: import matplotlib as mpl
         import matplotlib.pyplot as plt
         %matplotlib inline
         txt = '''Source: International Monetary Fund.'''
```

```
# Plot US share of world exports
combined.usshare.plot(grid=True, figsize=(9, 5), color="blue", linewidth=2,)
plt.ylabel('percentage of world exports')
plt.xlabel('Year')
plt.text(-50,3.8,txt)
plt.title('U.S. Share of World Exports');
```



The graph shows a decrease in the U.S. share of exports from nearly 20 percent in 1966 to roughly 9 percent in 2015. We can also easily examine how changes in the U.S. share of exports compare with changes in the share of Japan and China.

In [13]:
```
combshares = combined[['usshare', 'chinashare', 'japanshare']]
shares = list(combshares);

# Plot various shares of world exports
combined[shares].plot(grid=True, figsize=(9, 5))
plt.ylabel('percentage of world exports')
plt.xlabel('Year')
plt.text(-50,-3,txt)
plt.title('Share of World Exports');
```



5 Export dataset to .csv

Let's save the dataset in a portable format that can be read by any statistical software. My preference is to create a .csv file, which I can use for my U.S. Macroeconomic and Markets Dashboard, for example.

```
In [14]: combined.to_csv('us_share_exports.csv')
```

In closing, I've found that the time investment in programming machine access to data is paid off by the time savings in retrieving updated data. IMF data offers excellent country coverage and a high degree of reliability. With very little code, economists can access the latest IMF data and incorporate it directly into their work.