Paper 90-2019

Connecting to Datasets through Python and SAS®

Joe Matise, NORC at the University of Chicago

ABSTRACT

Python is a powerful tool for working with data, and makes the perfect partner to SAS to process data, particularly data that is obtained from online sources where programmers have developed open source Python code to connect to published APIs.

SAS has developed two tools to make working with Python easier for SAS programmers, without needing to know nearly any Python - and also made working with SAS easier for Python programmers who do not know SAS! In this paper, we explore these tools, as well as how to use SAS with Jupyter Notebooks, and show how to download some data from a few data sources.

This paper is aimed at an intermediate level programmer, but does not require particular SAS or Python knowledge and should be comprehensible to anyone. A basic understanding of using APIs to access data can be helpful but is not required.

INTRODUCTION

In today's era of open data, it is easier than ever to connect to large, complex public datasets, courtesy of the vast trove of open source software on sites like Github. While SAS tends to be underrepresented in the open source world, Python is one of the most common languages that open source packages are developed in.

At the same time, SAS is one of the most powerful tools available for quickly producing analyses, reports, and crunching large amounts of data. While Python is capable of doing most things SAS is, it is often easier to use SAS for the same task, and in many environments SAS servers are far more powerful.

Fortunately for SAS developers and Python developers alike, it is now possible to develop in SAS and Python while knowing primarily only one language or the other. SAS provides two packages, SASPy and the saskernel for Jupyter notebooks, which allow Python developers to write Python code that will call SAS routines – or allow SAS developers to use Python code to query datasets and then move the data easily into SAS for further processing there.

In this paper we will begin by explaing how to install Python and the SASPy interface module, as well as discussing installing and running Jupyter Notebooks. We will then explore the topic of open datasets, how to use already existing Python tools to connect to them, and walk through an example data pull using SASPy, Jupyter Notebooks, and of course, SAS.

STARTING OUT: INSTALLING PYTHON

Python is a powerful interpreted programming language which can be object-oriented, procedural, or functional, depending on the developer's preference. It is an open source language, supported by the Python Foundation, with an incredibly diverse array of open-source packages available for it. With the appropriate package, Python can be a very powerful statistical programming package rivalling SAS in capability and ease of use, though it has a much steeper learning curve.

To begin with, we will need to install Python on the machine you will work from. Python is an open source programming language that is supported by the Python Foundation, and like many open source languages has a number of packages you can choose to install depending on your needs. For this tutorial, we will use the Anaconda3 package.

Anaconda3 is a package of packages: it contains Python itself, as well as several packages for Python such as NumPy (useful for math and statistics work), SciPy (useful for scientific work), Pandas (a data manipulation package), and many more – over a hundred packages.

To download Anaconda, first install Condas (the Anaconda package manager), located at https://conda.io/docs/user-guide/install/download.html , and follow the instructions to install Condas and then Anaconda.

One thing to watch out for here: make sure to set the PATH variable in Windows to include your python root directory and \scripts subdirectory to it. This is important because it tells Windows where to find your python executable and your python scripts when you don't explicitly specify their locations.

INSTALLING OTHER ADD-ONS

The first thing you should familiarize yourself with is not python, but *pip*. Pip is the python package installer, and you can use it to install any packages you did not get with Anaconda. For most packages you can simply run (from the command line):

```
pip install <package>
```

Pip can either install from the Python Package Index (https://pypi.org/), its default, or from another location such as Github. In this paper we will use Github for the most part as typically it will be more up to date.

If you do want to use Github as the source for pip, you will also need to install git if you do not already have it installed (https://git-scm.com/downloads).

LET'S GET SASSY: INTRODUCING SASPY

SASPy is the Python package that will let us work with SAS in Python. To start with, install SASPy using pip:

```
pip install git+https://github.com/sassoftware/saspy
```

Or, you can use this if you did not install git:

```
pip install saspy
```

They should be the same version, but if they are different the Github version is typically newer.

Then, there are a few configuration steps. They are documented at https://sassoftware.github.io/saspy/install.html and reproduced here.

First, copy the file SASConfig.py to SASConfig_personal.py (these are all found in the package folder, which you can find by typing `pip show saspy`; for me it was installed in the folder C:\Programdata\Anaconda3\Lib\site-packages\saspy\). You will make some edits to this based on how you will connect to SAS.

CONNECTING TO A LOCAL SAS INSTALL

If you have SAS installed locally and are on Windows, then you will edit the SAS_config_names field to read:

```
SAS_config_names=['winlocal']
```

This tells SASPy to use the local installation.

CONNECTING TO A SERVER INSTALLATION

If you have a server installation, then you need to provide SASPy with the connection information to that. The full set of possibilities is listed in the above link; here is an example for one server configuration, which is a Linux Server using IOM. Your SAS Administrator should be able to give you the necessary details here. These are found at the bottom of the file, and you need to edit only the one that is relevant to your situation. Usually the 'iomhost' is the location of your IOM server, and the rest are often correct as-is.

```
winiomlinux = {'java'   : 'java',
               'iomhost'   : 'linux.iom.host',
               'iomport'   : 8591,
               'encoding'  : 'latin1',
               'classpath' : cpW
               }
```

If you use that, then you would also set:

```
SAS_config_names=[winiomlinux]
```

You also may need to add the path to `sspiauth.dll` to your system path in windows.

USING SASPY

Here's an example program using SASPy:

```
import saspy

sas = saspy.SASsession(cfgname='winlocal')
cars = sas.sasdata("CARS","SASHELP")
cars.describe()
```

Let's break it down and explain what each line does.

1. The Import Statement

```
import saspy
```

This statements tells Python to include the SASPy module. It allows us to use the various methods that are included in that module, namely the methods that interact with SAS.

2. The SAS connection

```
sas = saspy.SASsession(cfgname='winlocal')
```

This creates the connection with SAS, authenticating (if in a server environment) and spinning up a SAS session. Make sure the configuration name you used in the setup is the one in this method call.

3. Pulling down a SAS dataset

```
cars = sas.sasdata("CARS","SASHELP")
```

Here we create a python dataset (type, SASdata) which contains a dataset defined in SAS. The first argument is the dataset name, the second is the library name. This is the python equivalent of:

```
data cars;
    set sashelp.cars;
run;
```

For the most part, we can treat this just like a SAS dataset.

4. Running a SAS-like method

```
cars.describe()
```

Here we call the `describe` method of the `SASdata` class, which basically runs a PROC MEANS on the dataset.

```
>>> cars.describe()
```

	Variable	Label	N	NMiss	Median	...	Min	P25	P50	P75	Max
0	MSRP	NaN	428	0	27635.0	...	10280.0	20329.50	27635.0	39215.0	192465.0
1	Invoice	NaN	428	0	25294.5	...	9875.0	18851.00	25294.5	35732.5	173560.0
2	EngineSize	Engine Size (L)	428	0	3.0	...	1.3	2.35	3.0	3.9	8.3
3	Cylinders	NaN	426	2	6.0	...	3.0	4.00	6.0	6.0	12.0
4	Horsepower	NaN	428	0	210.0	...	73.0	165.00	210.0	255.0	500.0
5	MPG_City	MPG (City)	428	0	19.0	...	10.0	17.00	19.0	21.5	60.0
6	MPG_Highway	MPG (Highway)	428	0	26.0	...	12.0	24.00	26.0	29.0	66.0
7	Weight	Weight (LBS)	428	0	3474.5	...	1850.0	3103.00	3474.5	3978.5	7190.0
8	Wheelbase	Wheelbase (IN)	428	0	107.0	...	89.0	103.00	107.0	112.0	144.0
9	Length	Length (IN)	428	0	187.0	...	143.0	178.00	187.0	194.0	238.0

You see the variables, their labels, and some basic statistics about them. (There are a few more hidden by the '…', including Mean itself.) This is actually built similarly to a Pandas dataframe, for those python programmers out there; if you know Pandas you know what to do here.

Other Methods

SASPy offers many methods, which are documented at https://sassoftware.github.io/saspy/api.html (the API Reference). For the most part, they can be grouped as methods of the SASData object (such as `describe()` above), Procedure methods, and submitting SAS code directly through the submit() method. We recommend developers familiarize themselves with this documentation as there is a lot of power in SASPy beyond simply connecting SAS and Python.

SAS Procedures

Some SAS procedures have been implemented directly in SASPy. Browse the documentation to see if a procedure you need is there; and if not, there's a way to add one if you are feeling adventurous.

Submiting SAS code directly

So far, if you are a SAS programmer and not a Python programmer, you have likely been looking at this a bit askance – *I thought this was about helping SAS programmers run SAS in Python, not making me learn another language!* Well, this section is for you.

While you can do a lot in Python, if you're not a Python developer you may prefer to run things in SAS as much as possible. For that, the developers of SASPy created the `submit()` method. That takes one argument – a text string – and sends it up the wire to SAS, returning whatever it gets back as text (SAS results, basically). It also creates datasets which you can then access through the SASData class as above.

```
rc = sas.submit("""
proc contents data=sashelp.class;
run;
""")
```

The above returns the SAS results which can be parsed with the HTML method in the IPython.display module or other similar modules.

BLASTING OFF: INTRODUCING JUPYTER NOTEBOOKS

Jupyter notebooks, thus named because of the early support for JUlia, PYThon, and R, are a great way to develop code, particularly for doing data exploration or writing up analyses you want to distribute along with the code.

Jupyter notebooks combine text markup with runnable code blocks that display their results inline. The text is written in Markdown, an easy to learn syntax for writing nicely formatted text. Jupyter notebooks can be distributed to other programmers and run on their machines easily, allowing for reproducible results with minimal effort.

SAS added support for Jupyter notebooks, courtesy of the sas kernel, which you can install using pip:

```
pip install sas_kernel
```

Jupyter is installed as part of the Anaconda installation. Once you have installed the sas kernel, then open a command line window and run it:

```
jupyter notebook
```

That will run the jupyter notebook server, which will then load a web page which contains the notebooks available to you.

CREATING YOUR FIRST JUPYTER NOTEBOOK

To create a notebook, select New, and select SAS as the kernel to use. It opens a new window to the new notebook. You can give it a name by clicking on the "Untitled" at the top. You will see a single entry box, currently a Code block. The two blocks you will usually work with are Code blocks (which contain code to run) and Markdown blocks (which contain text).

Let's change the first cell to a Markdown block – specifically a header. Select the cell but don't click in the text box. Then type '1'. This will change it to a header (H1) markdown cell, like a SAS title. Go ahead and put a title there.

Then click shift+enter which will "run" the block (i.e., print the header out). It will also create a new code block underneath.

In that code block enter some SAS code. Say, run a proc on `SASHELP.CLASS` or similar. This is just normal SAS code, nothing different from entering it in a regular SAS session. Then press shift+enter, and see what happens. You should get after a few seconds a message that it connected to SAS, with a subprocess ID and then some SAS output based on the proc you ran.

LIMITATIONS (FOR NOW)

Jupyter notebooks do not, at least for the moment, support multiple kernels *in a single notebook*. While you can have Python notebooks and SAS notebooks, you can't directly use both in one notebook. You can of course have a Python notebook running SASPy, and from there submit SAS code, however.

INFORMATION WANTS TO BE FREE: OPEN DATASETS AND PYTHON

Researchers who depend on datasets produced by other organizations, such as governments or nonprofits, are undoubtedly familiar with the concept of Open Data. Open data refers to datasets that are freely provided for the public benefit, with no or minimal restrictions on use. Perhaps the most common example used in survey research in many countries are Census datasets; provided by many countries for use by their citizens, these datasets are commonly used to balance samples and provide weighting targets.

One of the biggest challenges to using open datasets is that there are often many steps required to get the data into the shape you need. Downloading it takes time, then learning the structure of the data and reformatting it to meet your needs takes even more.

Fortunately, for many open datasets someone has already done the work for you. This is where Python comes in.

PYTHON MODULES FOR OPEN DATA

Many major open datasets have one or more Python implementations to choose from, either provided by the data providers or more often provided by a kind-hearted soul who's taken the time to do all of the hard work These modules can usually be found on GitHub, or simply by searching the web for the name of the open dataset provider and python.

In this paper we will work with the US Census data, and a package developed to work with that data. The creatively named Census package connects to the US Census API and downloads data in a neat, compact format that is easy to work with and handles much of the back end data manipulation for you. All you need to know is the table you need and have an API key and you can get right to the fun part – working with your data.

HANDS ON DATA WITH THE US CENSUS

First, we will need to install the package that we found (on GitHub) locally. We use `pip` for that:

```
pip install git+https://github.com/datamade/census
```

Then, we need to obtain an API key. This is a common requirement, and typically free, but this allows the data provider to uniquely identify requests and implement rate limiting to prevent their servers being overloaded. That's available here:

- https://api.census.gov/data/key_signup.html

Finally, because the census API is based on tables, we need to look up what table we want to download data for. For this example we will query the ACS 5-year tables. There are several ways to look up what table you want, we used https://factfinder.census.gov/faces/nav/jsf/pages/searchresults.xhtm and searched "Sex by age", as we want to see a few breakdowns of the ACS 5 year totals. That is the B01001 table, so we'll query that.

Pulling the data with Python, then sending up to SAS

Here is the code we'll use to download the data and then send it to SAS. We'll present the code first and then break it down line by line.

The code:

```
from us import states

from census import Census

import pandas as pd

import saspy

#First, define a connection using our API key
c = Census("e7cd51b9bf4678c6e1c993116f0984b44cded26")


#Download a couple of census questions

cen_data =
pd.DataFrame(c.acs5.state(('NAME','B01001_002E','B01001_026E'),
Census.ALL))


#now, upload to SAS

upload_to_SAS(df=cen_data,path='c:\\temp',sastbl='uscen_data')
```

The import section

```
from us import states

from census import Census

import pandas as pd

import saspy
```

Here we import the python modules we will need. SASPy, of course, Pandas to create a data frame from the results, the Census module, and then the states module if we want to filter to only certain states.

Creating a connection to the Census

```
c = Census("e7cd51b9bf4678c6e1c993116f0984b44cded26")
```

This creates a connection to the Census server, and submits our API key. Here you will provide your own API key, not the fake one above, of course. Once you have run this, you can use the object `c` to make what calls you need to the Census API.

Get that data!

```
cen_data =
pd.DataFrame(c.acs5.state(('NAME','B01001_002E','B01001_026E'),
Census.ALL))
```

Unpacking this some, this creates a DataFrame named `cen_data` which contains the results of the call to the Census API. `C.acs5.state` is how we make that call; this specifies which survey to query (the ACS 5 year) and then to get data for a specific state or group of states. We then pass it the list of information we want, mostly table names, plus the geography we want – in this case, all states.

This returns data that can be coerced into a Pandas dataframe, and is stored in the variable `cen_data`.

Push the data up to SAS

```
upload_to_SAS(df=cen_data,path='c:\\temp',sastbl='uscen_data')
```

Now, we push the data to SAS. This line actually will not work on its own; that is because it is a user defined function that needs to be compiled first. Here is the definition:

```
def upload_to_SAS(df,path,sastbl):
    sas = saspy.SASsession(cfgname='winlocal')
    sas.saslib('demo',path=path)
    sas.df2sd(df=df,table=sastbl,libref='demo')
```

This creates a new SAS session, defines a SAS library with the path you provide, and then pushes a pandas dataframe (i.e., the one you just created) up to SAS.

One note here: the SAS session is not passed to or from the function, so it will create a new session each time you run it, and you cannot use the session outside of the function. The dataset of course will be available in whatever library you put it in, but that library will need to be re-created. If you want to be able to reuse the SAS session, a modification could be made to pass that as a parameter.

```
def upload_to_SAS(df,path,sastbl,sas):
    sas.saslib('demo',path=path)
    sas.df2sd(df=df,table=sastbl,libref='demo')
```

And then you call it like this:

```
upload_to_SAS(df=cen_data,path='c:\\temp',sastbl='uscen_data', sas=sas)
```

This requires you to have already defined the SAS connection before you call the function.

At this point, your dataset is now in SAS, and you can either continue working in SASPy using `sas.submit()`, or you can move into SAS and work from there.

DATA… I NEED MORE DATA!

Census data is just one example of open data that can be easily read in with a Python API. There are many other sources of data – for those Canadians out there StatCan has a similar API, for example, for downloading Canadian vital statistics and census-type information; see https://github.com/ianepreston/stats_can/ for an example Python module, and see https://www150.statcan.gc.ca/n1/en/type/data to identify the table numbers for StatCan.

Google BigQuery (https://cloud.google.com/bigquery/docs/reference/libraries) is a great option for downloading large datasets. Kaggle is another popular source of data, particularly for learning (or participating in Kaggle competitions), and has a easy to use API (https://github.com/Kaggle/kaggle-api).

One good option for finding datasets is the CKAN service. CKAN stores information (metadata) about open datasets, voluntarily uploaded, including how to access them. While CKAN does not actually store the data themselves, it is often easy to get to it from the information CKAN has. You can connect Python to CKAN using the CKAN API (https://github.com/ckan/ckanapi) or browse the readme on that page for more about how to use CKAN. Many of the datasets indexed by CKAN also have Python packages available to use for downloading their data as well.

CONCLUSION

A veteran SAS developer has many tools in their belt, and thanks to SAS Institute's support, Python is now one of the more powerful tools that a SAS programmer can utilize. Python's widespread adoption in the open source community has lead it to be a great source for code other developers have created to accomplish common tasks, so we do not have to reinvent the wheel to connect to many data sources. Even with only a rudimentary understanding of Python, we can take advantage of these open-source modules to save time and access more data than ever before.

We would encourage you to not only be a user, but a creator as well. If you do find modules have not yet been developed for your particular use case, make one and share it. SASPy also is an open source project, and has simple documentation for how to contribute a new module of your own. Contributing to projects is often as easy as writing code to solve your problem and then posting it to the project's Github page. And even if you don't feel comfortable writing code yourself to post, you can create an issue and include the necessary information to solve it, so someone else can easily add it to the project.

ACKNOWLEDGMENTS

Thanks to Matt Kastin for giving helpful feedback on the paper, and the whole NORC Data Services team for giving feedback on the presentation that goes with it. Thanks to Derek Montrichard and the Toronto SAS Users Group for generously inviting me to give the original talk that spawned this paper. Also, thanks to Dave Trevarthen and Jeff Vose for giving me much needed support and encouragement to write this paper and to present it.

RECOMMENDED READING

- *SASPy documentation,* https://sassoftware.github.io/saspy/
- *How to configure Python and SASPy,* http://www.scsug.org/wp-content/uploads/2018/10/McCarthy-How-to-configure-Python-and-SASPy.pdf
- *The SAS Dummy, How to run SAS programs in Jupyter Notebooks,* https://blogs.sas.com/content/sasdummy/2016/04/24/how-to-run-sas-programs-in-jupyter-notebook/

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Joe Matise
NORC at the University of Chicago
Email: matise.joe@gmail.com
https://stackoverflow.com/users/1623007/joe