Paper 3189-2019 (Submitted to SAS Global Forum)

Everything is better with friends: Executing SAS® code in Python scripts with SASPy

Isaiah Lankham, University of California, Office of the President, Oakland, CA

Matthew Slaughter, Kaiser Permanente Center for Health Research, Portland, OR

ABSTRACT

SASPy is a module developed by SAS Institute for the Python programming language, providing an alternative interface to the SAS System. With SASPy, SAS procedures can be executed in Python scripts using Python syntax, and data can be transferred between SAS datasets and their Python `DataFrame` equivalent. This allows SAS programmers to take advantage of the flexibility of Python for flow control, and Python programmers can incorporate SAS analytics into their scripts.

This paper provides several examples of common data-analysis tasks using both regular SAS code and SASPy within a Python script, highlighting important tradeoffs for each and emphasizing the value of being a polyglot programmer fluent in multiple languages. Instructions are also included for replicating these examples with the JupyterLab interface for SAS University Edition, which includes everything needed to try out SASPy.

Examples of SAS and Python working together like BFFs (Best Friends Forever) can also be downloaded as a Jupyter notebook file from https://github.com/saspy-bffs/sgf-2019-how

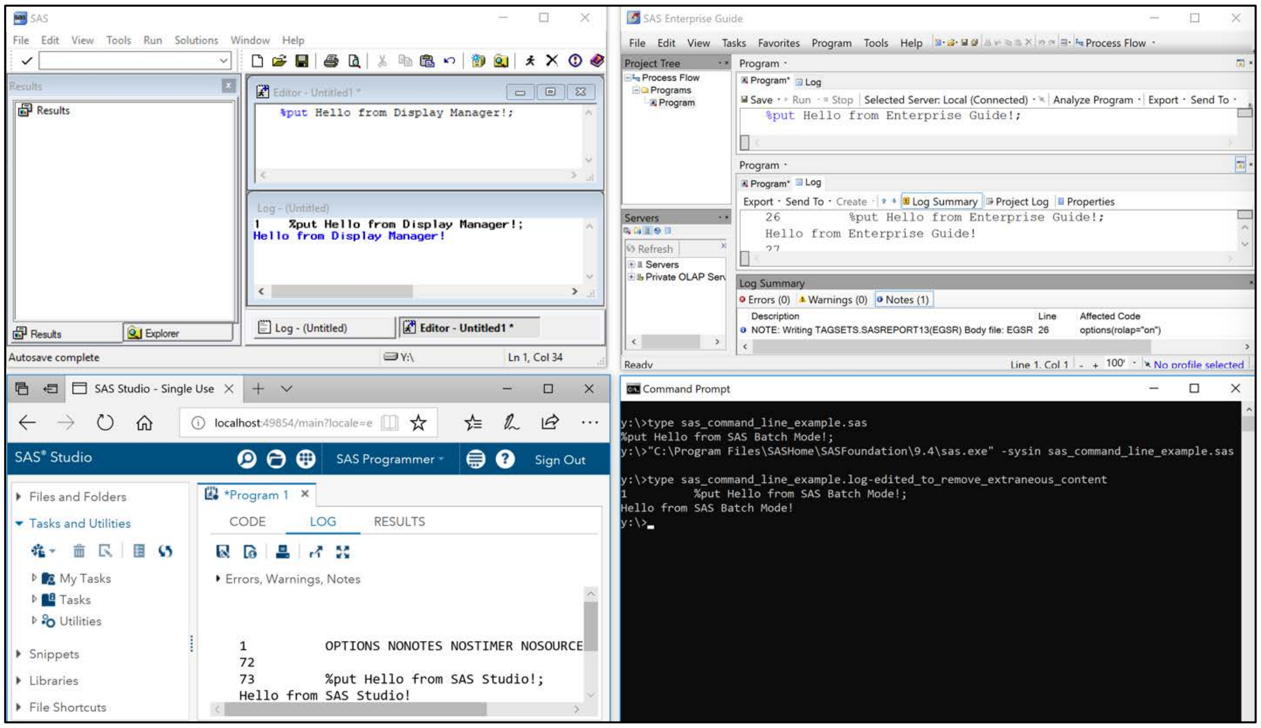
INTRODUCTION

SAS PROGRAMMING: ONE SYSTEM, MANY INTERFACES

Given a list of data-analysis tasks to perform, what SAS interface would you choose?

If you're a typical user of the SAS language, there's a good chance you'll default to the Display Manager (aka the SAS Windowing Environment) or Enterprise Guide®, which are two of the three integrated development environments (IDEs) included with Base SAS, and you might not even realize the web-based IDE SAS Studio is included [44]. Base SAS users also have the option of writing SAS code in a text editor (e.g., Notepad, which ships with the Windows operating system) and submitting programs in Batch Mode from the command line, which is a fourth, non-IDE option (see Figure 1). And even if you choose the completely separate product SAS University Edition, you'll still need to decide between SAS Studio and yet another web-based IDE called JupyterLab (see Appendix A).

The SAS System supports many interfaces because of its MultiVendor Architecture™ (MVA), which separates code creation from code execution [1]. Whether you program in the Display Manager's Enhanced Editor or Notepad, you're still submitting code to a SAS kernel, which is a standalone program taking SAS language statements as input and returning two values: (1) A log describing how the code was executed, and (2) the code execution's results, which are typically some form of output in text or HTML format. IDEs obscure this distinction by bundling together a text editor, related tooling, and the ability to submit code to a kernel. In addition, good SAS IDEs automatically display the log and code-execution results, enabling development to become a seamless feedback loop, whether connected to a local or remote SAS kernel.



```
%put Hello from Display Manager!;
```

```
1    %put Hello from Display Manager!;
Hello from Display Manager!
```

```
%put Hello from Enterprise Guide!;
```

```
26         %put Hello from Enterprise Guide!;
Hello from Enterprise Guide!
27
```

Description	Line	Affected Code
NOTE: Writing TAGSETS.SASREPORT13(EGSR) Body file: EGSR	26	options(rolap="on")

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE
72
73         %put Hello from SAS Studio!;
Hello from SAS Studio!
```

```
y:\>type sas_command_line_example.sas
%put Hello from SAS Batch Mode!;
y:\>"C:\Program Files\SASHome\SASFoundation\9.4\sas.exe" -sysin sas_command_line_example.sas

y:\>type sas_command_line_example.log-edited_to_remove_extraneous_content
1          %put Hello from SAS Batch Mode!;
Hello from SAS Batch Mode!
y:\>_
```

Figure 1. In clockwise order, starting from the bottom-left, the four main SAS interfaces shipped with SAS System Version 9 are SAS Studio, Display Manager, Enterprise Guide, and SAS Batch Mode (with command-line tools use to print the contents of the input file and resulting log file).

In summary, SAS's MVA enables you to choose between many different ways of writing and executing SAS code, each having its own tradeoffs. However, borrowing from [10], IDE-based SAS interfaces also are simultaneously "WYSIWYG — what you see is what you get" and "WYSIAYG—what you see is *all* you get." The opposite extreme is a non-IDE interface like Batch Processing at the command line, which is more complex but also significantly more malleable since a nearly unlimited number of command-line tools can be combined together. Given the ever-increasing ubiquity and flexibility of the Python programming language [7], we consider the "best of both worlds" SAS interface provided by the Python module SASPy to be somewhere in the middle.

SASPy is a Python module developed by SAS Institute as an interface for the SAS System [18], enabling Python scripts to connect to a SAS kernel (see Section 1) and load SAS dataset files into their Python equivalent, which are `DataFrame` objects provided by the pandas module (see Section 2). In addition, convenience functions can be used to invoke SAS procedures directly on SAS datasets with Python syntax (see Section 3), and SAS code can also be programmatically generated and submitted to a SAS Kernel, enabling Python to serve as a surprisingly powerful replacement for the SAS Macro Facility (see Section 4).

Even though they're sometimes viewed as competitors, SAS and Python both have their advantages, so choosing between them can be more a matter of preference and convenience. But with SASPy, there's no reason to see SAS and Python as anything less than complementary tools (or even BFFs, best friends forever). As you'll see, SAS often provides a more direct path for many data-analysis tasks, while Python is often more straightforward for control flow and dataset manipulation. For each example, some variation of the SAS's MEANS procedure will be used.

As background, Python is an open-source language originally developed in the 1990s for teaching programming [47]. Highly praised for its straightforward syntax, which resembles DATA step programming in SAS, Python initially became popular as a "glue" language [48] and is now frequently referred to in the Python community as the "second best language" for everything from data science to web development. Many popular websites are Python applications, including Disqus, Dropbox, Instagram, Pinterest, Reddit , Spotify, and Uber [5]. There are also many success stories attributed to Python. Perhaps the most famous is YouTube, which outpaced its now-defunct rival Google Video in feature development and was eventually acquired by Google. Per [6], YouTube's 20 developers relied on Python, whereas Google Video's hundreds of developers used C++.

SAS PROGRAMMING: ONE SYSTEM, MANY LANGUAGES

If using Python to create and submit code to a SAS kernel seems strange, think about this: There's a good chance you're already doing the exact opposite every time you use SAS!

Because of its MVA, SAS code can be written in a mixture of programming languages and language dialects, each of which the SAS kernel either understands natively or farms out to a different kernel to execute on its behalf. In addition to the usual DATA step and SAS macro language code, the SAS System can natively understand each of the following:

- the object-oriented DS2 (think "DATA step 2") language within PROC DS2 [26]
- the Graph Template Language (GTL) within PROCs SGRENDER and TEMPLATE [21]
- the vector-based Interactive Matrix Language (IML) within PROC IML [29]

Non-proprietary languages supported by the SAS System include the following:

- C/C++ within PROC PROTO [32]
- Groovy (a Java-like language) within PROC GROOVY [20]
- Java (and other languages[1]) via DATA step Java objects [40]
- Lua within PROC LUA [28]
- Perl-like regular expressions within *prx*-prefixed functions and call routines [30]
- R within PROC IML [39]
- Structured Query Language (SQL) within PROC SQL [41] and PROC FEDSQL [27]
- Table Producing Language (TPL) within PROC TABULATE [49]

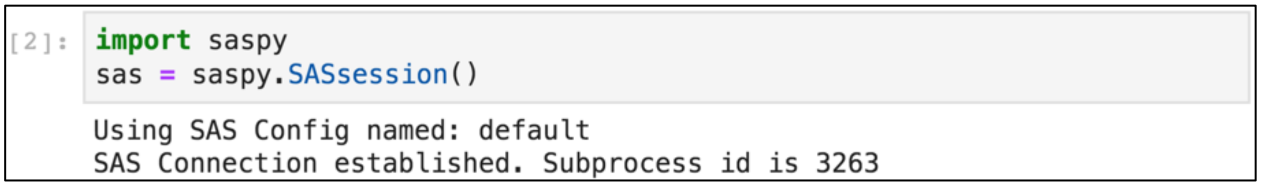
In other words, SAS users already need to be polyglot programmers capable of working in multiple languages simultaneously. Borrowing a term from philosophy, this means SAS programming is inherently *syncretic* in nature, blending together multiple ways of problem solving, which will ideally become more than the sum of its parts. We have great flexibility in creating and executing code, and using the SAS System to its full potential often requires a confluence of many complementary ways of thinking.

For the purposes of this paper, all screenshots are from the JupyterLab interface for SAS University Edition, which comes pre-configured with SASPy (see Appendix A). However, SASPy can also be used outside of SAS University Edition, per instructions at [24].

Similar introductory papers for SASPy include [8], [13], and [17]. Papers using SASPy as a tool include [3], [12], [42], and [43].

As a starting point in using Python for data-science applications, we highly recommend the freely available, concise, and comprehensive overview *A Whirlwind Tour of Python* [46].

[1] Per [33], Java Objects can also be used to execute Python code within a DATA step. If combined with SASPy, SAS code conceivably could be used to invoke Python code, which itself could invoke SAS code, and so on. Whether this has any practical applications (other than the obvious practical joke of Python code calling the SAS code that called it, creating an infinite loop) is left as an exercise to the reader.



```
[2]: import saspy
     sas = saspy.SASsession()

     Using SAS Config named: default
     SAS Connection established. Subprocess id is 3263
```

Figure 2. A connection to a SAS session is established from a Python notebook in JupyterLab using SASPy.

SECTION 1: USING SASPY TO CONNECT TO THE SAS SYSTEM

Within Python (e.g., using a Python notebook in SAS University Edition, per Appendix A, or setting up stand-alone Python and SAS installations and then installing/configuring SASPy, per [24]), we can establish a connection to SAS as follows:

```
# Python code for Figure 2
import saspy ❶
sas = saspy.SASsession() ❷
```

The **import** statement in ❶ loads the SASPy module, providing access to its methods and objects in subsequent statements. The assignment in ❷ uses dot notation, invoking the `SASsession` method (included in the `saspy` module) and establishes a connection to a SAS session, which is called sas for convenience (see Figure 2). In all subsequent lines of code within the same Python file, we can now use sas to execute SAS code or operate on SAS datasets. We can also get the full SAS session log at any point using print(`sas.saslog()`).

GETTING INFORMATION ABOUT THE SAS KERNEL

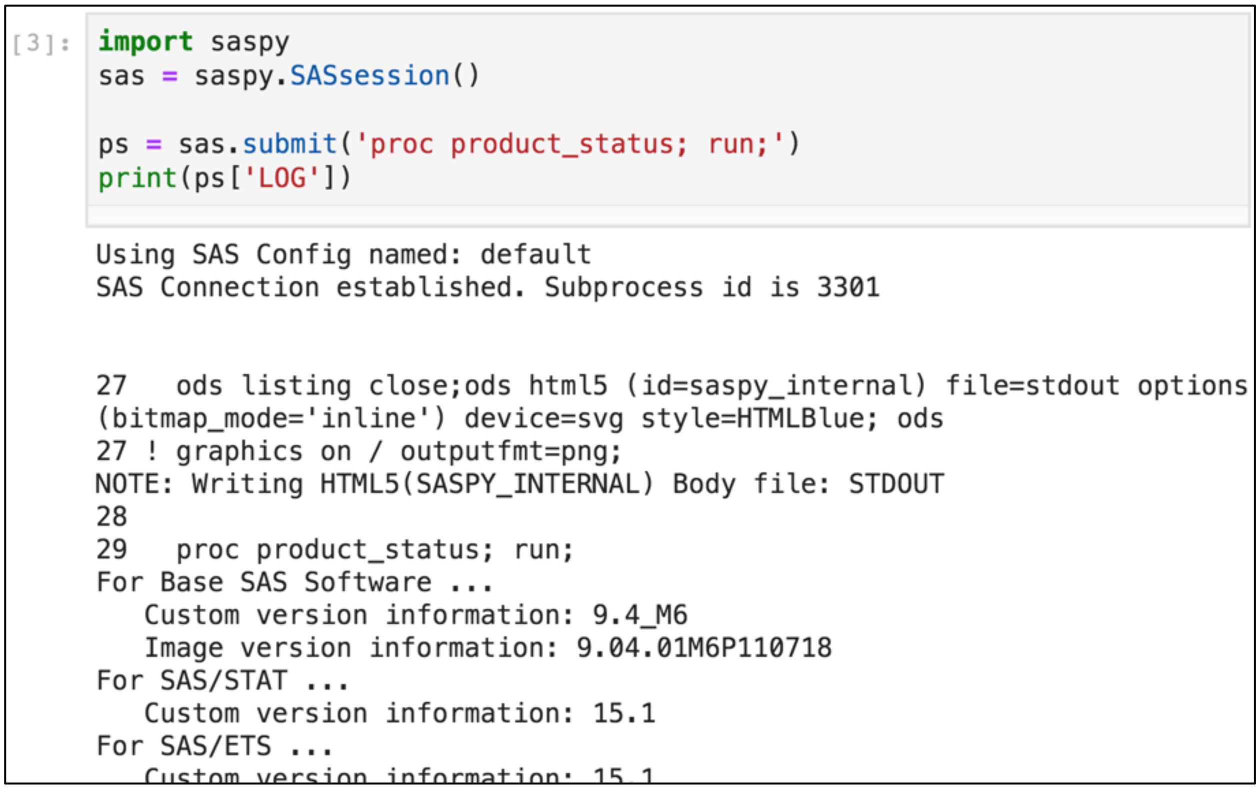
Since SASPy works by establishing a connection to an existing SAS installation, whether on the local machine or a remote server, it provides access to (and is limited to) the SAS components licensed and installed. To explore the components available from Python, we can view the results of submitting the PRODUCT_STATUS procedure [31] as follows:

```
# Python code for Figure 3
ps = sas.submit('proc product_status; run;') ❶
print(ps['LOG']) ❷
```

The assignment in ❶ creates a new Python dictionary called ps, which is the result of the object sas (created in the previous example) calling its `submit` method to execute the SAS code in quote marks. Dictionaries are one of the most fundamental data structures in Python, being the analog of SAS formats and DATA step hash tables, and are more generally called *associative arrays* or *maps* because they associate *keys* with *values*. In this case, the dictionary ps has the following key-value pairs, with the keys appearing in the brackets on the left-hand sides of the equal signs and their associated values on the right-hand sides:

- ps['LOG'] = '<the log resulting from submitting PROC PRODUCT_STATUS>'
- ps['LST'] = '<the results of submitting PROC PRODUCT_STATUS>'

The Python function print is used in ❷ to print the log returned by submitting PROC PRODUCT_STATUS, which is accessed using bracket notation to extract the value associated with key 'LOG' (see Figure 3).



```
[3]: import saspy
     sas = saspy.SASsession()

     ps = sas.submit('proc product_status; run;')
     print(ps['LOG'])
```

```
Using SAS Config named: default
SAS Connection established. Subprocess id is 3301


27   ods listing close;ods html5 (id=saspy_internal) file=stdout options
(bitmap_mode='inline') device=svg style=HTMLBlue; ods
27 ! graphics on / outputfmt=png;
NOTE: Writing HTML5(SASPY_INTERNAL) Body file: STDOUT
28
29   proc product_status; run;
For Base SAS Software ...
   Custom version information: 9.4_M6
   Image version information: 9.04.01M6P110718
For SAS/STAT ...
   Custom version information: 15.1
For SAS/ETS ...
   Custom version information: 15.1
```

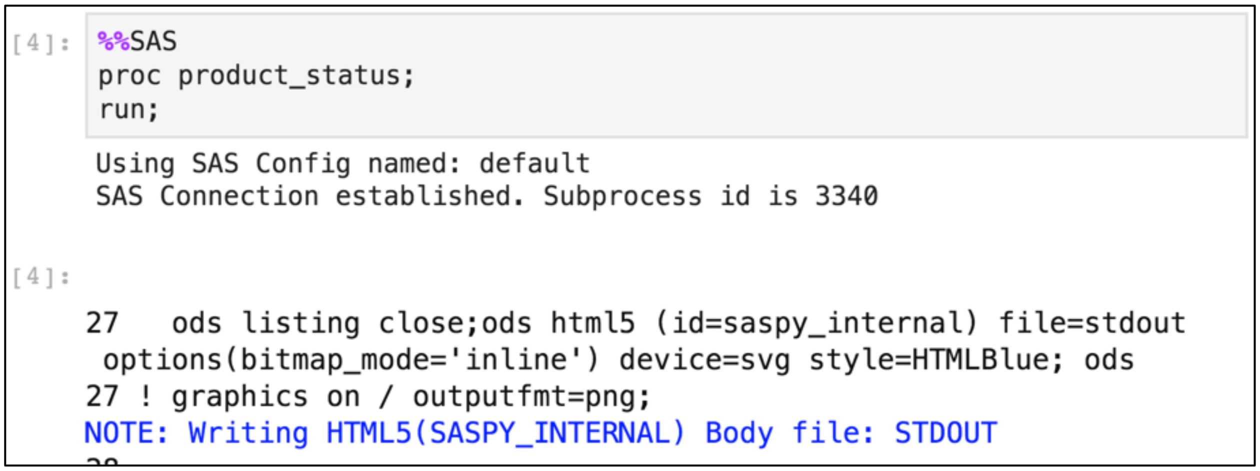
Figure 3. PROC PRODUCT_STATUS is submitted to the SAS kernel included in SAS University Edition from a Python notebook in JupyterLab using SASPy. All printed SAS components (and their associated procedures) are available in SASPy.

A useful alternative to the submit method is the %%SAS magic command, which SASPy makes available when it's imported. Magic commands are Jupyter-specific meta-commands that appear at the start of a cell and modify how the rest of the cell's contents are executed, as in the following example:

```
# Python code (with JupyterLab magic command %%SAS) for Figure 4
%%SAS ❶
proc product_status; ❷
run; ❷
```

The %%SAS magic command in ❶ causes all subsequent cell contents (❷) to be submitted directly to the SAS kernel associated with SASPy when it was imported, rather than be interpreted as Python code, but will still be color-coded as Python syntax. The results (or log, if no results are generated or an error occurs) will then be displayed (see Figure 4).

In other words, %%SAS is a convenient way of invoking SAS in the middle of a Python notebook, and it can also be made available with the command %load_ext saspy.sas_magic if SASPy has not already been imported, where %load_ext is a standard Python magic command for loading language extensions like other magic commands [11]. However, since % is also a SAS macro trigger, this could potentially cause confusion unless clearly used in the context of a Python notebook with SASPy acting as a bridge to a SAS kernel, and where it's clear that all subsequent cell contents should be read as SAS code. In addition, as an important caveat, any % in subsequent lines after %%SAS will be passed directly to the SAS kernel and interpreted as SAS macro calls [35].



```
[4]: %%SAS
     proc product_status;
     run;

     Using SAS Config named: default
     SAS Connection established. Subprocess id is 3340

[4]:

    27   ods listing close;ods html5 (id=saspy_internal) file=stdout
     options(bitmap_mode='inline') device=svg style=HTMLBlue; ods
    27 ! graphics on / outputfmt=png;
    NOTE: Writing HTML5(SASPY_INTERNAL) Body file: STDOUT
```

Figure 4. PROC PRODUCT_STATUS is submitted to the SAS kernel included in SAS University Edition from a Python notebook in JupyterLab using SASPy with the %%SAS magic command. Note also that log output is color coded, unlike in Figure 3.

GETTING INFORMATION ABOUT THE PYTHON ENVIRONMENT

As an aside, we can also get version and package information for Python as follows:

```
# Python code for Figure 5
import platform ❶
print(platform.sys.version) ❷
help('modules') ❸
```

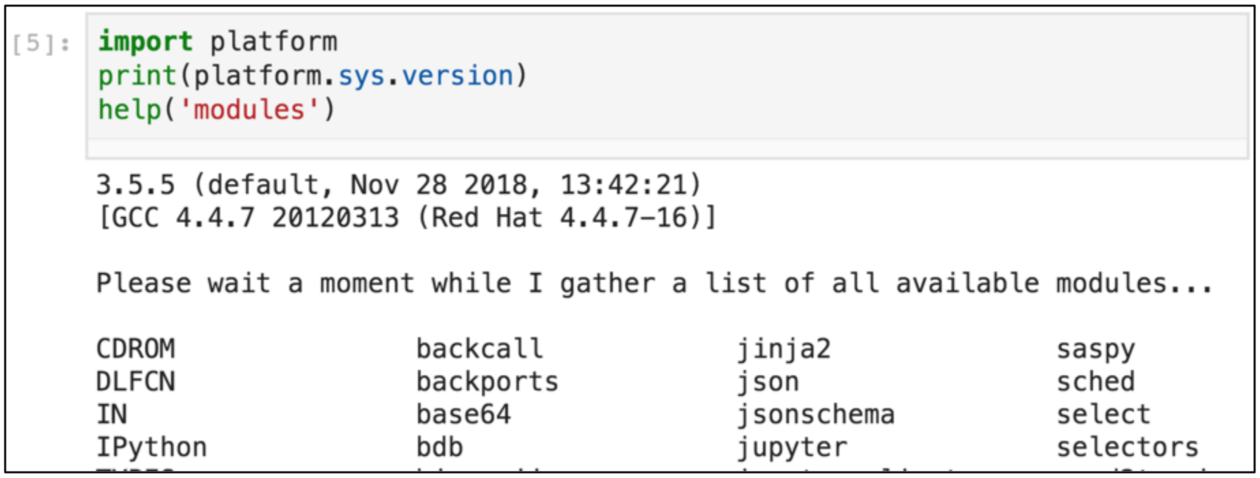
The **import** statement loads a new module in ❶. The `platform` module is then used in ❷ by invoking the sub-module object sys nested inside of it, and sys invokes the object version nested inside of it (think Russian nesting dolls or turduckens). A list of all installed modules available to be imported is then printed using the Python print function in ❸ (see Figure 5).

Just as SASPy is limited by installed (and licensed) SAS components, Python is limited by the modules that have been explicitly imported from the list produced by help('modules'). In keeping with the philosophy of being a "batteries included" language, the many modules in Python's standard library provide ready-made solutions for a wide variety of programming tasks[2]. In addition, numerous third-party modules are actively developed and made freely available through sites like GitHub (https://github.com/) and the Python Package Index (https://pypi.org/), which makes the chances of finding an appropriate module for a specific task even greater.

CAUTION: PYTHON AND SAS SYNTAX DIFFERENCES

Before proceeding, we'd like to pause and point out some important differences between SAS and Python syntax, which might not be obvious to programmers whose prior experience is limited to SAS. (More complete overviews of Python syntax can be found by searching popular reference sites, like *Learn X in Y minutes* [4].)

[2] In keeping with the spirit of Python having been named after the British comedy troop Monty Python, there are also quite a few Easter eggs in the standard library. For example, `import antigravity` will open a new web browser window to the XKCD comic https://xkcd.com/353/. Unfortunately, though, this functionality isn't available in SAS University Edition due to its server-client configuration.



```
[5]: import platform
     print(platform.sys.version)
     help('modules')

     3.5.5 (default, Nov 28 2018, 13:42:21)
     [GCC 4.4.7 20120313 (Red Hat 4.4.7-16)]

     Please wait a moment while I gather a list of all available modules...

     CDROM                backcall             jinja2               saspy
     DLFCN                backports            json                 sched
     IN                   base64               jsonschema           select
     IPython              bdb                  jupyter              selectors
```

Figure 5. Python version information, and a list of available modules, are printing for a Python notebook in JupyterLab using SAS University Edition. Note the following about this output: (1) Python is not up-to-date, with the most release being 3.7.3, as of this writing, and (2) the list includes both standard-library modules (e.g., `json` for handling JSON files) and third-party modules (e.g., `jupyter`, which is part of the code behind the Python-powered JupyterLab).

Note the following:

- Python is case sensitive, with `IMPORT PLATFORM` different from **import** `platform`.
- Semicolons are not required after each line of code in Python. Instead, they are typically only used to separate multiple statements placed on the same line; e.g., the previous example could have instead been written on one line as follows:

  ```
  import platform; print(platform.sys.version); help('modules')
  ```

  However, this style is generally discouraged since it lowers readability.
- Python has multiple, interchangeable options for creating quoted strings. Single and double quotes have identical behavior (with help('modules') and help("modules") having the exact same effect), and three repeated quote marks can be used to create either single-line strings (with help('modules') and help('''modules''') having the exact same effect) or strings with embedded line breaks (see Figure 10).
- Python has distinct assignment (=) and test-for-equality (==) operators. For example, the Python equivalent of the SAS DATA step code

  ```
  if today() = '04JUL19'd then fireworks = 'Yes!';
  ```

  is as follows:

  ```
  import datetime ❶
  if datetime.date.today() == datetime.date(2019,7,4): ❷
     fireworks = 'Yes!' ❸
  else: ❹
     fireworks = '' ❺
  ```

  The **import** statement loads the `datetime` module in ❶, and we check whether today's date is the fourth of July in ❷. If so, we set `fireworks` to 'Yes!' in ❸. Otherwise, we set `fireworks` to the empty string in ❹–❺. Note that unindenting ❸ or ❺ would produce an error since indentation is used to determine scope in Python.

SECTION 2: USING SASPY TO ACCESS SAS DATASETS

Now that we can connect to a SAS kernel, we explore how SASPy can be used to load a SAS dataset (i.e., a file with extension .sas7bdat) into its Python equivalent, which is the DataFrame object provided by the third-party pandas module.

LOADING A SAS DATASET INTO A PANDAS DATAFRAME

We can use SASPy to load a SAS dataset as follows:

```
# Python code for Figure 6
fish_df = sas.sasdata2dataframe(table='fish',libref='sashelp') ❶
fish_df.describe() ❷
```

The assignment statement in ❶ creates a DataFrame object called fish_df and copies the contents (but not the metadata) of SAS dataset sashelp.fish into it using the sasdata2dataframe method of sas (created in a previous example). In other words, fish_df is the Python analog of a SAS dataset. (The concept of "data frame" is shared with the language R. However, whereas R's data frames are a built-in type, Python's definition is provided by the extremely popular third-party module pandas [16]. For a good overview, see [15].) The values in fish_df are then summarized using the describe method, which is the Python analog of SAS's PROC MEANS (see Figure 6).

Just like a SAS dataset, fish_df is essentially a two-dimensional array of values with rows representing observations and columns representing properties of observations. However, whereas SAS datasets are kept on disk and processed row-by-row (as needed), fish_df lives entirely in memory, allowing random access to all values simultaneously. This greatly increases the operations possible, but as a trade-off, the size of fish_df cannot exceed system memory[3], whereas SAS datasets can be arbitrarily large. Once a DataFrame has been defined, we can use this random access for on-the-fly reshaping like the following:

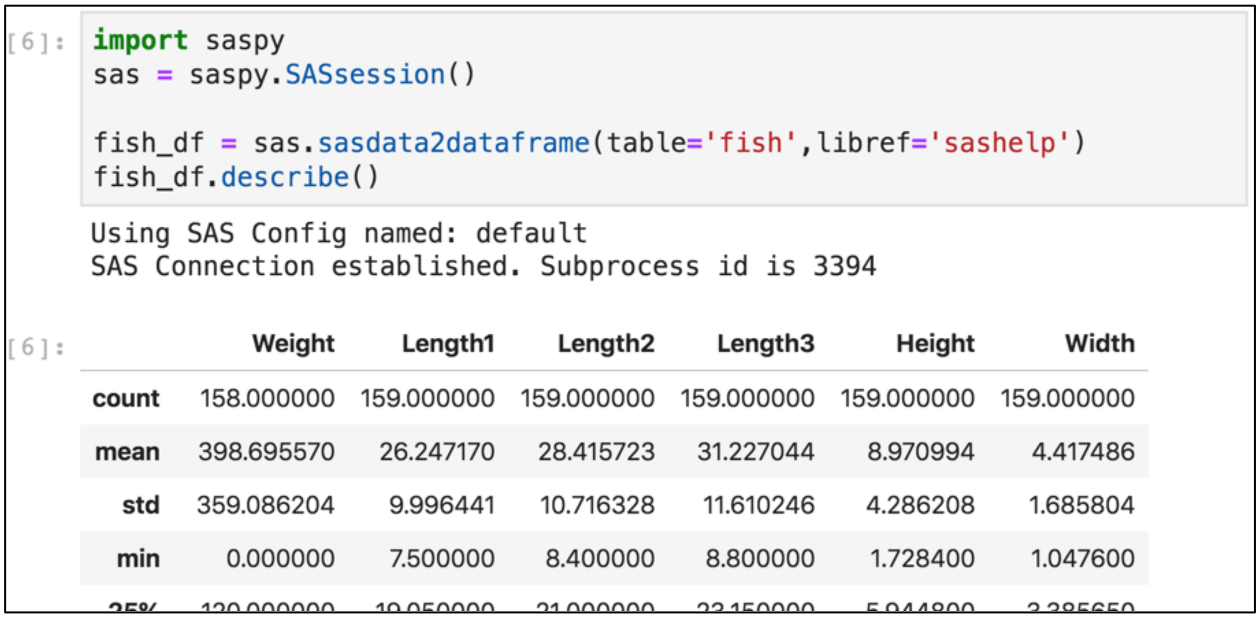
```
# Python code for Figure 7
fish_df_g   = fish_df.groupby('Species') ❶
fish_df_gs  = fish_df_g['Weight'] ❷
fish_df_gsa = fish_df_gs.agg(['count', 'std', 'mean', 'min', 'max']) ❸
print(fish_df_gsa) ❹
```

The assignment statement in ❶ creates a new DataFrame called fish_df_g using the groupby method to group rows in fish_df (created in the previous example) by values in the column 'Species'. Similarly, fish_df_gs is created in ❷ by extracting the 'Weight' column from fish_df_g using bracket notation, and fish_df_gsa is created in ❸ using the agg method to aggregate the rows of fish_df_gs using each of the five functions in the list ['count', 'std', 'mean', 'min', 'max']. The contents of fish_df_gsa are then printed using the print function in ❹ (see Figure 7).

This sequence of operations can also be succinctly combined into a "one-liner" as follows:

```
# Python code for Figure 7 as a one-liner
fish_df_gsa = fish_df.groupby('Species')['Weight'].agg(
   ['count', 'std', 'mean', 'min', 'max']
)
print(fish_df_gsa)
```

[3] The author of pandas recommends having 5-10 times as much RAM as the size of a dataset [14]. However, this limitation can be addressed with in-memory optimization or by switching to disk-based storage similar to SAS datasets, per [50].

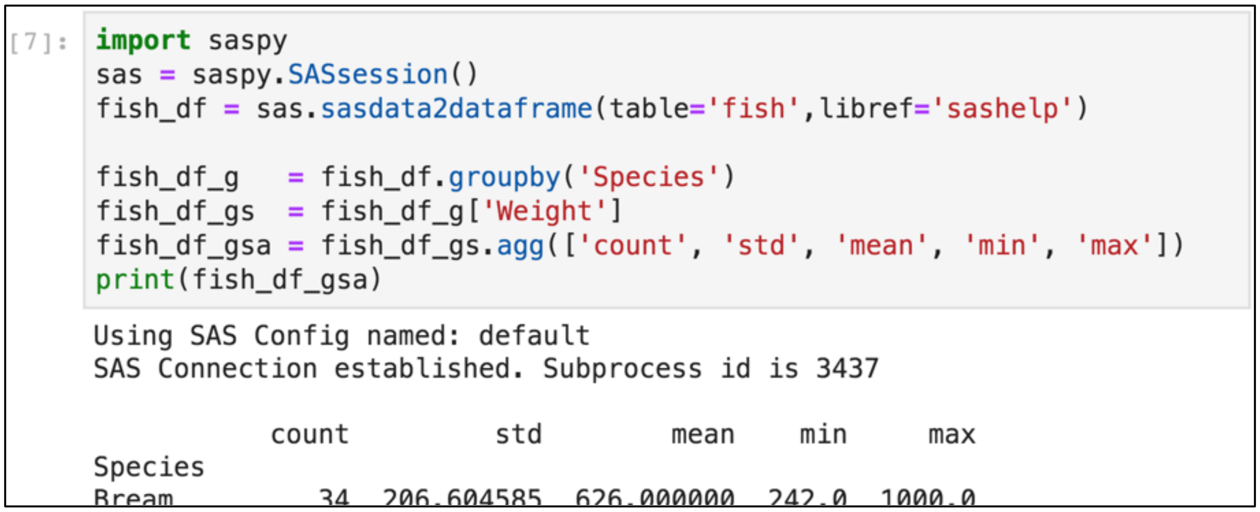


```
[6]: import saspy
     sas = saspy.SASsession()

     fish_df = sas.sasdata2dataframe(table='fish',libref='sashelp')
     fish_df.describe()

     Using SAS Config named: default
     SAS Connection established. Subprocess id is 3394
```

[6]:

	Weight	Length1	Length2	Length3	Height	Width
count	158.000000	159.000000	159.000000	159.000000	159.000000	159.000000
mean	398.695570	26.247170	28.415723	31.227044	8.970994	4.417486
std	359.086204	9.996441	10.716328	11.610246	4.286208	1.685804
min	0.000000	7.500000	8.400000	8.800000	1.728400	1.047600
25%	120.000000	19.050000	21.000000	23.150000	5.944800	3.385650

Figure 6. The results of the `describe` method applied to the pandas `DataFrame` `fish_df` from a Python notebook in JupyterLab, with SASPy used to load the contents of SAS dataset `sashelp.fish` into `fish_df`.



```
[7]: import saspy
     sas = saspy.SASsession()
     fish_df = sas.sasdata2dataframe(table='fish',libref='sashelp')

     fish_df_g   = fish_df.groupby('Species')
     fish_df_gs  = fish_df_g['Weight']
     fish_df_gsa = fish_df_gs.agg(['count', 'std', 'mean', 'min', 'max'])
     print(fish_df_gsa)

     Using SAS Config named: default
     SAS Connection established. Subprocess id is 3437

                count         std        mean    min     max
     Species
     Bream         34  206.604585  626.000000  242.0  1000.0
```

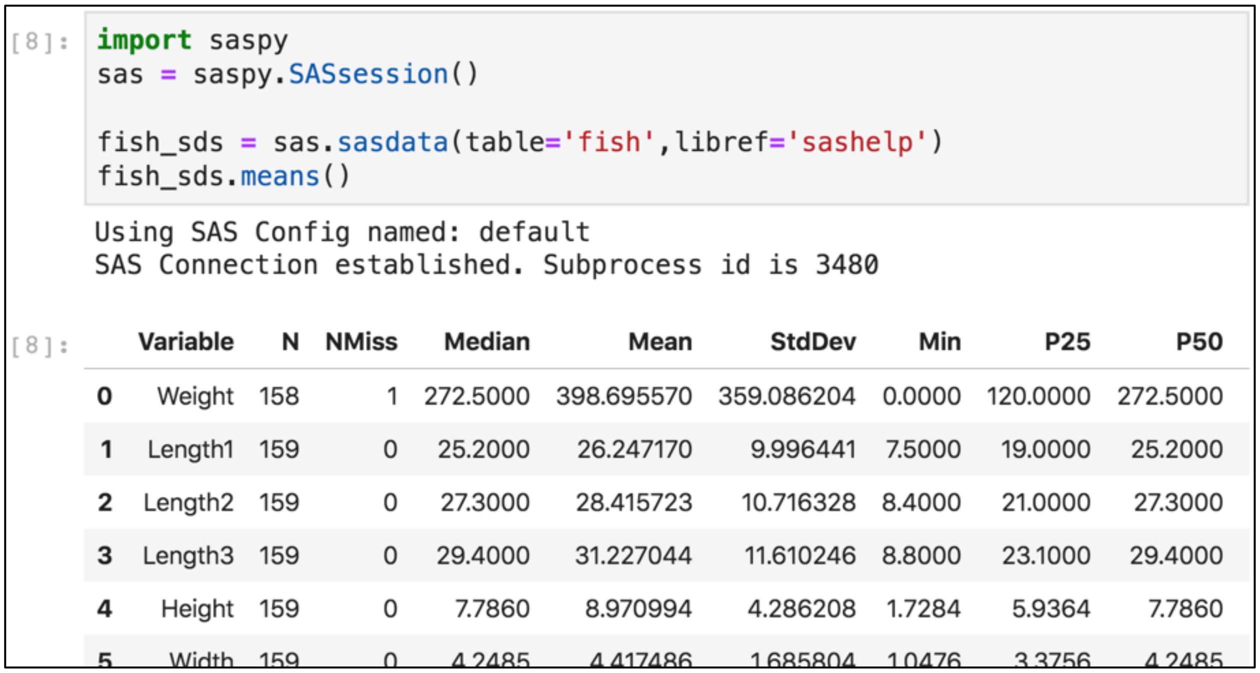
Figure 7. The results of on-the-fly reshaping of the pandas `DataFrame` `fish_df` from a Python notebook in JupyterLab.

LOADING A PANDAS DATAFRAME INTO A SAS DATASETS

Finally, to complete the round-trip back to SAS, we can convert to a SAS dataset as follows:

```
sas.dataframe2sasdata(fish_df_gsa,table='fish_sds_gsa',libref='work')
```

In other words, the `dataframe2sasdata` method of `sas` (created in a previous example) is used to create a new SAS dataset file `fish_sds_gsa.sas7bdat` in SAS's Work library.



```
[8]: import saspy
     sas = saspy.SASsession()

     fish_sds = sas.sasdata(table='fish',libref='sashelp')
     fish_sds.means()

     Using SAS Config named: default
     SAS Connection established. Subprocess id is 3480
```

[8]:

	Variable	N	NMiss	Median	Mean	StdDev	Min	P25	P50
0	Weight	158	1	272.5000	398.695570	359.086204	0.0000	120.0000	272.5000
1	Length1	159	0	25.2000	26.247170	9.996441	7.5000	19.0000	25.2000
2	Length2	159	0	27.3000	28.415723	10.716328	8.4000	21.0000	27.3000
3	Length3	159	0	29.4000	31.227044	11.610246	8.8000	23.1000	29.4000
4	Height	159	0	7.7860	8.970994	4.286208	1.7284	5.9364	7.7860
5	Width	159	0	4.2485	4.417486	1.685804	1.0476	3.3756	4.2485

Figure 8. The results of the means convenience method applied to SAS dataset sashelp.fish from a Python notebook in JupyterLab using SASPy. Even though summary statistics are being computed using SAS's PROC MEANS, the results are formatted using pandas-styled output.

SECTION 3: USING SASPY CONVENIENCE METHODS

Having seen how to import SAS datasets into Python by converting to a `DataFrame`, we now explore the complementary option of using SASPy to directly interacting with SAS datasets.

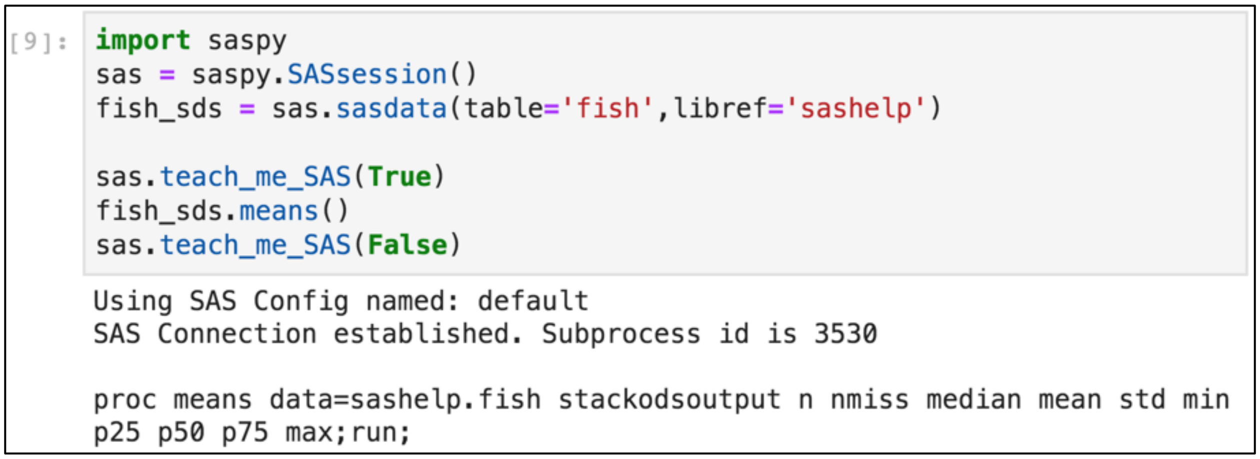
DIRECTLY OPERATING ON A SAS DATASET WITH CONVENIENCE METHODS

SASPy can directly operate on a SAS dataset as in the following example:

```
# Python code for Figure 8
fish_sds = sas.sasdata(table='fish',libref='sashelp') ❶
fish_sds.means() ❷
```

The assignment statement in ❶ creates a direct connection to the SAS dataset `sashelp.fish` using the sasdata method of the sas object (created in a previous example). In other words, `fish_sds` is effectively just a pointer to the physical file where `sashelp.fish` is stored (i.e., the file named `fish.sas7bdat` in SAS's SASHELP library). The contents of `sashelp.fish` are then summarized using the means convenience method in ❷, which implicitly invokes PROC MEANS (see Figure 8). In order words, `sashelp.fish` is read from disk and processed row-by-row, just like when PROC MEANS is used directly in SAS as follows:

```
* SAS code equivalent for Figure 8;
proc means
        data= sashelp.fish
        stackodsoutput n nmiss median mean std min p25 p50 p75 max
    ;
run;
```



```
[9]: import saspy
     sas = saspy.SASsession()
     fish_sds = sas.sasdata(table='fish',libref='sashelp')

     sas.teach_me_SAS(True)
     fish_sds.means()
     sas.teach_me_SAS(False)

     Using SAS Config named: default
     SAS Connection established. Subprocess id is 3530

     proc means data=sashelp.fish stackodsoutput n nmiss median mean std min
     p25 p50 p75 max;run;
```

Figure 9. The results of the "Teach Me SAS" sandwich applied to the `means` convenience method applied to SAS dataset `sashelp.fish` from a Python notebook in JupyterLab using SASPy. This is useful for Python programmers looking to learn SAS, as well as for experienced SAS users looking to modify the generated code.

This has the following tradeoffs:

- On the one hand, the Python syntax is more concise and flexible. For example, once `fish_sds` is defined, we could sequentially apply additional convenience methods like

  ```
  fish_sds.scatter(x='Weight',y='Height',title='Scatterplot Example')
  ```

  to obtain a scatter plot by implicitly invoking the SGPLOT procedure. In addition, the `sasdata` method can be used just as easily with SAS views as with physical files.

- On the other hand, our options are limited to convenience methods provided by SASPy, as opposed to the rich language available for manipulating a pandas `DataFrame` created with the `sasdata2dataframe` method. However, as SASPy is expanded to include additional convenience methods, more options will become available. And since SASPy is open source software, virtually anyone can create a GitHub account and contribute additional functionality.

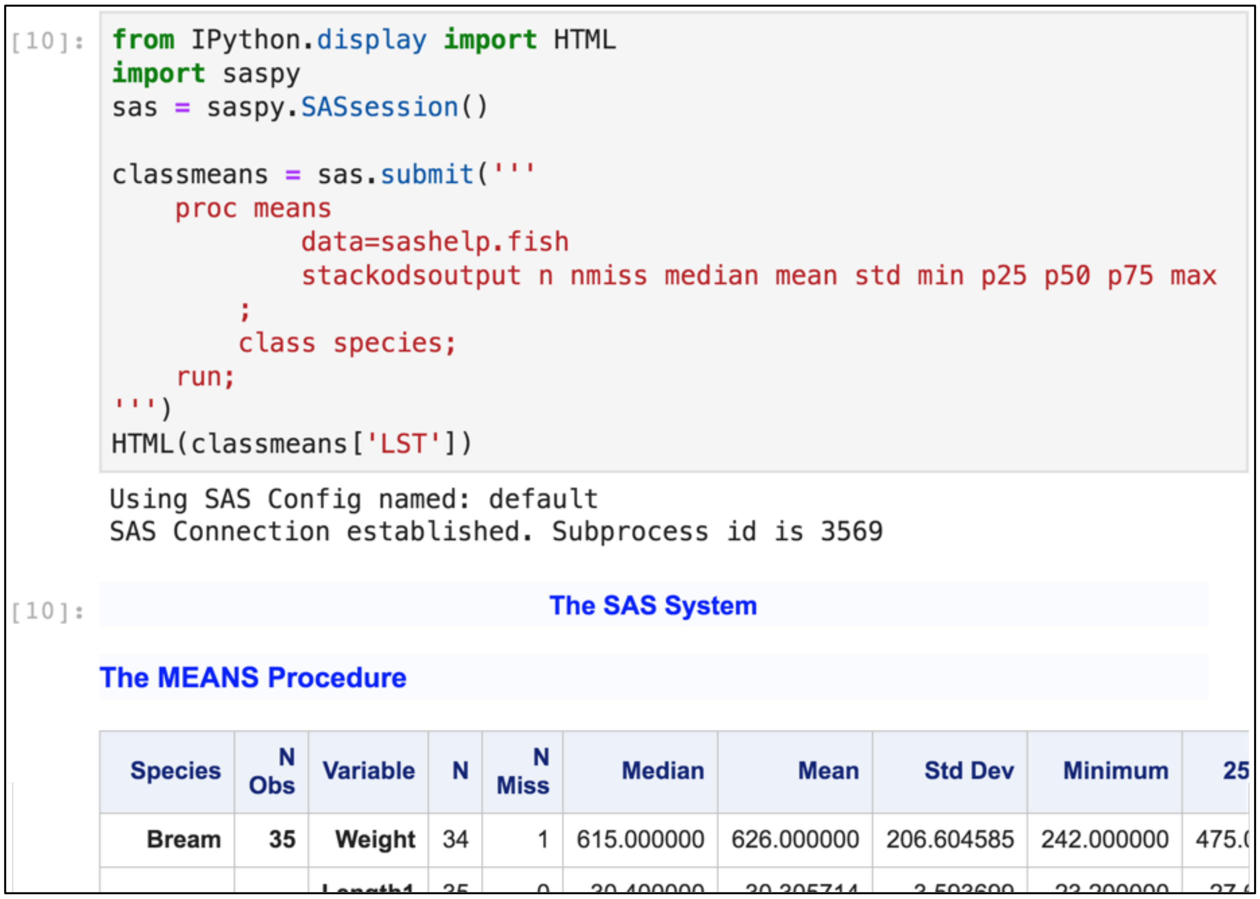
PRINTING SAS CODE GENERATED BY A CONVENIENCE METHOD

The SAS source code generated by SASPy convenience methods can be viewed as follows:

```
# Python code for Figure 9
sas.teach_me_SAS(True) ❶
fish_sds.means() ❷
sas.teach_me_SAS(False) ❸
```

The `teach_me_SAS` method of the sas object (created in a previous example) in ❶ takes a Boolean argument, meaning either the Python object **True** (equivalent to the integer 1) or the Python object **False** (equivalent to the integer 0). When passed **True**, all subsequent use of SASPy convenience methods like `fish.means()` in ❷ will display the code they generate, rather than executing it (see Figure 9). Then, when the "Teach Me SAS" sandwich is closed by passing **False** to `sas.teach_me_SAS` in ❸, normal execution of SASPy convenience methods is resumed for all subsequent lines of code.

The "Teach Me SAS" sandwich can be thought of as the SASPy analog of the "ODS sandwich" for opening and closing output destinations in SAS programming.



```
[10]: from IPython.display import HTML
      import saspy
      sas = saspy.SASsession()

      classmeans = sas.submit('''
          proc means
                  data=sashelp.fish
                  stackodsoutput n nmiss median mean std min p25 p50 p75 max
              ;
              class species;
          run;
      ''')
      HTML(classmeans['LST'])

      Using SAS Config named: default
      SAS Connection established. Subprocess id is 3569
```

[10]:

The SAS System

The MEANS Procedure

Species	N Obs	Variable	N	N Miss	Median	Mean	Std Dev	Minimum	25
Bream	**35**	**Weight**	34	1	615.000000	626.000000	206.604585	242.000000	475.
		Length1	35	0	30.400000	30.305714	3.503600	23.200000	27.

Figure 10. The results of directly submitting SAS code to a SAS kernel and then rendering the returned HTML from a Python notebook in JupyterLab using SASPy. Note the familiar HTMLBLUE-styled output.

The `teach_me_SAS` method can also be used to generate SAS code as a starting point for customized results. For example, since the `means` convenience method doesn't support the CLASS statement for PROC MEANS, we could add it ourselves as follows:

```
# Python code for Figure 10
classmeans = sas.submit(''' ❶
   proc means ❷
         data=sashelp.fish ❷
         stackodsoutput n nmiss median mean std min p25 p50 p75 max ❷
      ; ❷
      class species; ❸
   run; ❹
''') ❺
from IPython.display import HTML ❻
HTML(classmeans['LST']) ❼
```

The triple-quotes in ❶ and ❺ allow a Python string to run across multiple lines. The CLASS statement in ❸ then produces the statistics requested in ❷ grouped by values of species. As before, the `submit` method of the `sas` object (created in a previous example) returns a dictionary with key 'LST' corresponding to PROC MEANS's results as HTML output. The `HTML` function imported in ❻ is then used to render these results in ❼ (see Figure 10).

SECTION 4: USING SASPY TO IMITATE THE SAS MACRO FACILITY

Building on the examples in the previous two sections, we're now ready to illustrate how Python can be used to imitate the SAS Macro Facility as follows:

```
# Python code for Figure 11
from IPython.display import HTML ❶
sas_code_fragment = 'proc means data=sashelp.%s; run;' ❷
for dsn in ['fish','iris']: ❸
   display(HTML(sas.submit(sas_code_fragment%dsn)['LST'])) ❹
```

Lines ❶–❹ collectively accomplish the following:

- The relative **import** statement in ❶ makes the HTML method available.
- A string object named sas_code_fragment is created in ❷, which uses the C-style templating placeholder %s to denote where additional strings can be substituted in later uses of sas_code_fragment.
- A for-loop is then used to iterate over the two string values in list ['fish','iris'], meaning that the body of the loop (here, the single indented line ❹ since indentation is used to determine scope) will be executed for each value in the list, in order.
- During each iteration of the for-loop, the IPython method display is used to display rendered HTML output resulting from
    - o submitting the value of sas_code_fragment to a SAS kernel, but with either 'fish' or 'iris' used in place of %s, and
    - o extracting the results from the value returned by the submit method of the sas object (created in a previous example).

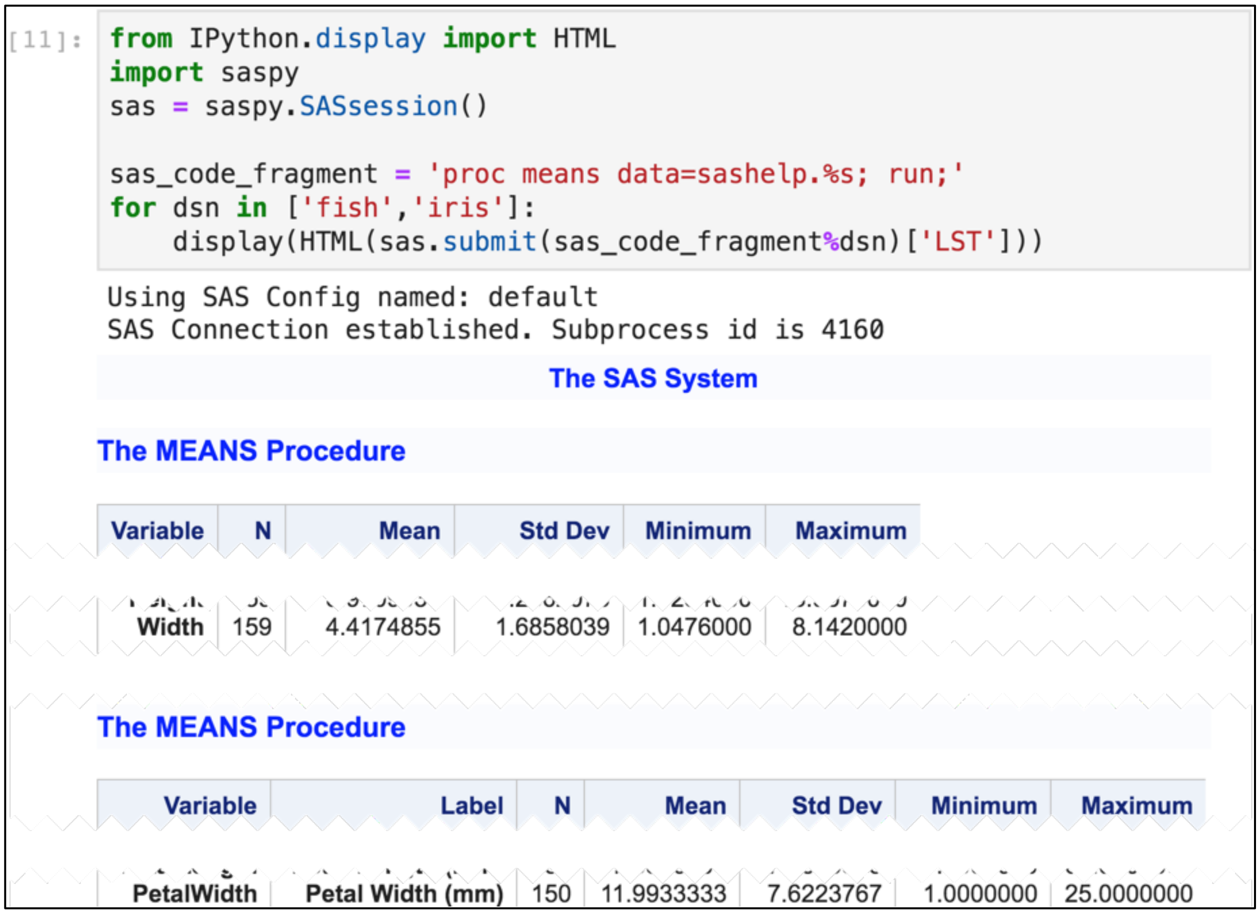
  As before, the submit method in ❹ returns a dictionary object with key 'LST' corresponding to the HTML results returned by PROC MEANS.

The net result is to output PROC MEANS applied to both sashelp.fish and sashelp.iris (see Figure 11). In other words, we've generated and submitted the following two lines of SAS code to the SAS kernel, and then we've rendered the resulting HTML output:

```
* SAS code equivalent for Figure 11, without macros;
proc means data=sashelp.fish; run;
proc means data=sashelp.iris; run;
```

The same outcome could also be achieved directly in SAS with the following code:

```
* SAS code equivalent for Figure 11, with macros;
%macro loop();
    %let dsn_list = fish iris;
    %do i = 1 %to 2;
        %let dsn = %scan(&dsn_list.,&i.);
        proc means data=sashelp.&dsn.;
        run;
    %end;
%mend;
%loop()
```



```
[11]: from IPython.display import HTML
      import saspy
      sas = saspy.SASsession()

      sas_code_fragment = 'proc means data=sashelp.%s; run;'
      for dsn in ['fish','iris']:
          display(HTML(sas.submit(sas_code_fragment%dsn)['LST']))
```

```
Using SAS Config named: default
SAS Connection established. Subprocess id is 4160
```

The SAS System

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Width	159	4.4174855	1.6858039	1.0476000	8.1420000

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
PetalWidth	Petal Width (mm)	150	11.9933333	7.6223767	1.0000000	25.0000000

Figure 11. The results of using Python to generating and submitting SAS code to a SAS kernel, and then rendering the returned HTML, from a Python notebook in JupyterLab using SASPy. Note the familiar HTMLBLUE-styled output.

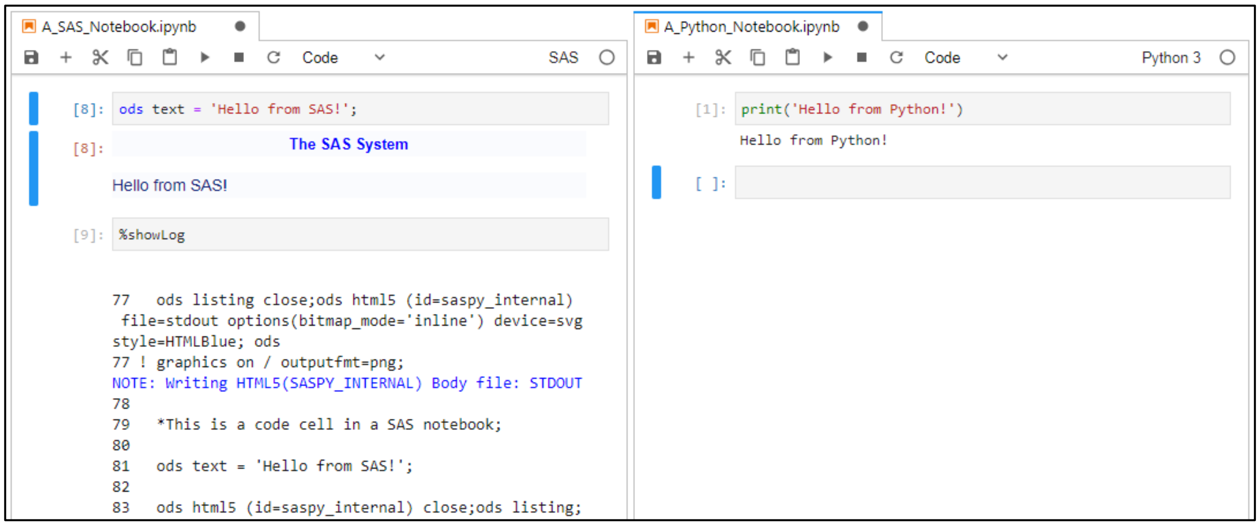
However, note the following differences:

- Python allows us to concisely repeat an arbitrary block of code by iterating over a list of values using a for-loop.
- The SAS Macro Facility, on the other hand, only provides do-loops based on integer-valued index variables like `i`, so clever tricks like the implicitly defined array `dsn_list` are need, together with functions like `%scan` to extract values.

APPENDIX A. GETTING STARTED WITH SAS UNIVERSITY EDITION AND THE JUPYTERLAB INTERFACE

WHAT IS SAS UNIVERSITY EDITION, AND HOW IS IT INSTALLED?

SAS University Edition (SAS UE) is a free version of SAS, which is aimed primarily at academic users but made available to the wider public for noncommercial use [38]. SAS UE includes many popular SAS programming products and two web-based interfaces, the SAS Studio IDE and JupyterLab, along with a Python installation that has SASPy installed and configured to use the included SAS kernel [19].



```
[8]: ods text = 'Hello from SAS!';
[8]:
                The SAS System

Hello from SAS!

[9]: %showLog

77   ods listing close;ods html5 (id=saspy_internal)
 file=stdout options(bitmap_mode='inline') device=svg
style=HTMLBlue; ods
77 ! graphics on / outputfmt=png;
NOTE: Writing HTML5(SASPY_INTERNAL) Body file: STDOUT
78
79   *This is a code cell in a SAS notebook;
80
81   ods text = 'Hello from SAS!';
82
83   ods html5 (id=saspy_internal) close;ods listing;
```

```
[1]: print('Hello from Python!')
     Hello from Python!

[ ]:
```

Figure 12. SAS and Python notebooks each display results obtained by executing code cells using their respective kernels, with JupyterLab tabs arranged to show the two notebooks side-by-side. The `%showLog` magic command is also used in the left-hand notebook to display the log for the SAS session.

As of this writing, SAS UE can be installed locally on Linux, macOS, and Windows using virtualization software (e.g., Oracle VirtualBox; https://www.virtualbox.org/), or it can be used with a cloud service like AWS [22]. This makes SAS UE available on a wide variety of platforms since most (but not all) modern computers either directly support virtualization or can be made to support it by updating BIOS settings [25].

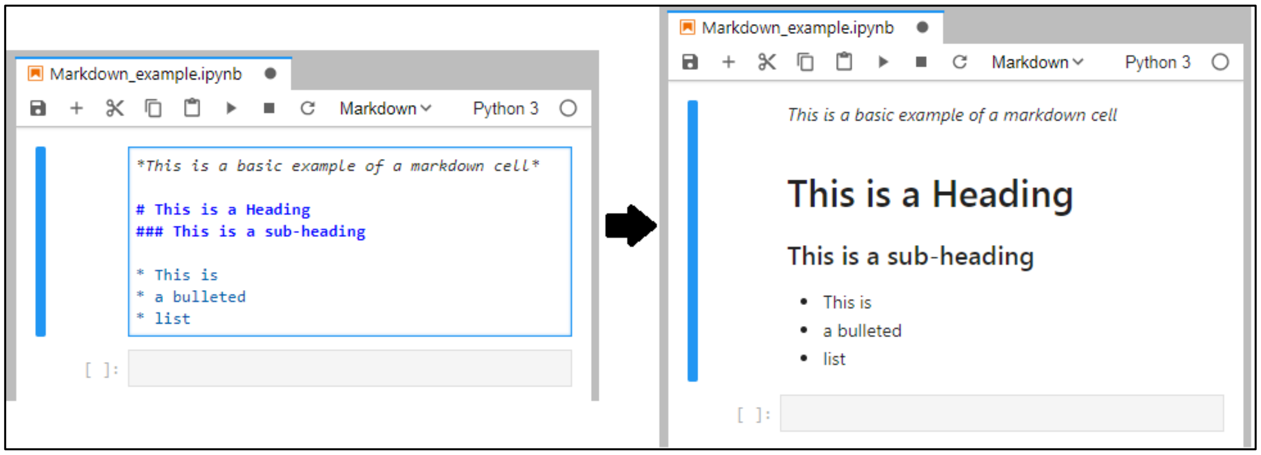
WHAT IS JUPYTERLAB?

JupyterLab is a browser-based development environment with a notebook interface supporting a wide variety of languages, including Python and SAS. Rather than requiring code, logs, and output to be saved in separate documents and subsequently synthesized to assemble a coherent narrative, JupyterLab encourages programmers to construct *computational narratives*, with documentation, code, and results intermixed. This approach reduces cognitive load and project-completion time, while enhancing reproducibility [9, 45]. The growing number of kernels supported by JupyterLab makes it an excellent choice for polyglot programming, whether used as part of SAS UE or as a stand-alone product.

Similar to SAS Studio, the browser-based JupyterLab interface can be configured to submit code to a SAS kernel, either on the local machine or on a remote server. In the case of SAS University Edition, JupyterLab has been preconfigured to use its local SAS kernel.

JUPYTER NOTEBOOK COMPONENTS

Jupyter notebooks are JSON files with extension `.ipynb` (short for IPython Notebook, which Jupyter is based on) that are organized as a series of cells. Each cell contains either formatted text (called Markdown; see below) or executable code, and code cells can be evaluated individually or in groups. In addition, cells can be merged, split, or moved around, as needed. When a code cell is executed, its contents are passed to the kernel associated with the notebook. The kernel then executes the code and returns any requested output. In addition, the results of the last line in the cell are automatically displayed. When submitting SAS code to a SAS kernel, its results (typically as rendered HTML5 output) are displayed (as in Figure 12), unless no output is created or an error occurs. In these cases, portions of the log are displayed instead [34].



```
*This is a basic example of a markdown cell*

# This is a Heading
### This is a sub-heading

* This is
* a bulleted
* list
```

This is a basic example of a markdown cell

This is a Heading

This is a sub-heading

- This is
- a bulleted
- list

Figure 13. A Markdown cell in a Python notebook before and after being rendered.

The magic command `%showLog` can also be used to view pertinent portions of the SAS log generated by the last cell executed in a SAS notebook (see Figure 12), and `%showFullLog` can be used to view the entire log of the current session with the SAS kernel [23].

Markdown cells are used to create formatted text cells [2]. Unlike Microsoft Word, where formatting is applied through a point-and click interface, Markdown is a plaintext markup language using special syntax to indicate how text should be rendered (see Figure 13). By intermixing Markdown cells with code cells, narrative context can be provided for the analysis contained in the notebook.

ADDITIONAL FEATURES AND UTILITIES

JupyterLab also includes a file browser and console, as well as tabs for viewing all currently running kernels, a list of available commands, and other metadata. The list of commands can be viewed by selecting the command tab, or with the keyboard shortcut `Ctrl+Shift+C` (or `Cmd+Shift+C` on macOS). Some commands are available at all times (such as `Shift+Enter` for executing a code cell or rendering a Markdown cell), but others are only available in command mode, which is activated by pressing the Escape key when a cell is actively being edited. (The Enter key can be used to return to editing mode).

CONCLUSION

A great strength of the SAS language is its ability to seamlessly integrate with other languages, including C/C++, Java, R, and SQL. SASPy expands this spirit of syncretic programming by integrating SAS with Python, allowing SAS programmers to enhance SAS analytics. The examples in this paper illustrated a small fraction of the potential use cases for SASPy, which can also interface with SAS components like SAS Enterprise Miner for predictive analytics [18]. Other options for interfacing SAS with Python include the SWAT package for the SAS Viya® platform [37] and the SAS Pipefitter project, which provides a Python API for building complex machine-learning pipelines [36].

With all of these exciting developments, there has never been a better time to be a multilingual programmer fluent in both SAS and Python. And with SASPy's ongoing development, the potential for using SAS with open-source software can only increase.

REFERENCES

[1] Cates, Mark. (1995) "Multivendor Architecture -- Supporting Feature Rich Platforms with a Uniformly Architected System." *Proceedings of the SAS Users Group International Conference,* Orlando, FL. Available at http://www.sascommunity.org/sugi/SUGI95/Sugi-95-222%20Cates.pdf

[2] Cone, Matt. *Markdown Guide*. Date accessed: 28MAR2019. Available at https://www.markdownguide.org/

[3] De Capite, Donna. (2018) "Docker Toolkit for Data Scientists — How to Start Doing Data Science in Minutes!" *Proceedings of the SAS Global Forum 2018 Conference*, Denver, CO. Available at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1875-2018.pdf

[4] Dinh, Louie. "Where X=python3." *Learn X in Y minutes*. Date accessed: 28MAR2019. Available at https://learnxinyminutes.com/docs/python3/

[5] Django Stars. (2019) "Top Seven Apps Built with Python." *Hacker Noon*. Date accessed: 28MAR2019. Available at https://hackernoon.com/top-seven-apps-built-with-python-2cd8dfd3c00a

[6] Driscoll, Mike. (2018) *Python Interviews: Discussions with Python Experts*. Packt Publishing: Birmingham, U.K.

[7] The Economist. (2018) "Python is becoming the world's most popular coding language." *Daily Chart*. Date accessed: 28MAR2019. Available at https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language

[8] Foreman, Carrie. (2018) "SAS and Python: The Perfect Partners in Crime." *Proceedings of the SAS Global Forum 2018 Conference*, Denver, CO. Available at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2597-2018.pdf

[9] Glanz, Hunter. (2017) "Revolutionizing Statistical Computing in SAS with the Jupyter Notebook." *Proceedings of the SAS Global Forum 2017 Conference*, Orlando, FL. Available at http://support.sas.com/resources/papers/proceedings17/0838-2017.pdf

[10] Hunt, Andrew and Thomas, David. (1999) *The Pragmatic Programmer: From Journeyman to Master*. Addison Wesley: Reading, MA.

[11] IPython Development Team. "IPython extensions." *IPython Documentation*. Date accessed: 28MAR2019. Available at https://ipython.readthedocs.io/en/stable/

[12] Islam, Tuba. (2017) "Open Your Mind: Use Cases for SAS and Open-Source Analytics." *Proceedings of the SAS Global Forum 2017 Conference*, Orlando, FL. Available at http://support.sas.com/resources/papers/proceedings17/SAS0747-2017.pdf

[13] McCarthy, Michael. (2018) "How to configure Python and SASPy." *Proceedings of the South Central SAS Users Group 2018 Educational Forum*, Austin, TX. Available at http://www.scsug.org/wp-content/uploads/2018/10/McCarthy-How-to-configure-Python-and-SASPy.pdf

[14] McKinney, Wes. (2017) *Apache Arrow and the '10 Things I Hate About pandas.'* Date accessed: 28MAR2019. Available at http://wesmckinney.com/blog/apache-arrow-pandas-internals/

[15] NumFOCUS (2018) "10 Minutes to pandas." *pandas: powerful Python data analysis toolkit*. Date accessed: 28MAR2019. Available at https://pandas.pydata.org/pandas-docs/stable/10min.html

[16] NumFOCUS. "pandas: powerful Python data analysis toolkit." *pandas: Python Data Analysis Library*. Date accessed: 28MAR2019. Available at http://pandas.pydata.org/pandas-docs/stable/

[17] Phillips, Jason. (2018) "A Basic Introduction to SASPy and Jupyter Notebooks." *Proceedings of the SAS Global Forum 2018 Conference*, Denver, CO. Available at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2822-2018.pdf

[18] SAS Institute. "A Python interface to MVA SAS." *Open Source from SAS Software*. Date accessed: 28MAR2019. Available at https://github.com/sassoftware/saspy

[19] SAS Institute. "Features." *SAS University Edition*. Date accessed: 28MAR2019. Available at https://www.sas.com/en_us/software/university-edition.html

[20] SAS Institute. "GROOVY Procedure." *Base SAS 9.4 Procedures Guide, Seventh Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=proc&docsetTarget=n12njw2j3tuptnn1bmmr0cl857d2.htm&docsetVersion=9.4&locale=en

[21] SAS Institute. "GTL and the Output Delivery System (ODS)." *SAS 9.4 Graph Template Language: Reference, Fifth Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=grstatgraph&docsetTarget=p0891gx3y0z8xqn1k9ijhv5xughi.htm&docsetVersion=9.4&locale=en

[22] SAS Institute. "How do I run SAS University Edition on Amazon Web Services Marketplace?" *SAS University Edition: Help Center*. Date accessed: 28MAR2019. Available at https://support.sas.com/software/products/university-edition/faq/AWS_runvApp.htm

[23] SAS Institute. "How do I view my SAS log in a Jupyter notebook?" *SAS University Edition: Help Center*. Date accessed: 28MAR2019. Available at https://support.sas.com/software/products/university-edition/faq/jn_viewSASlog.htm

[24] SAS Institute. "Installation and configuration." *SASPy*. Date accessed: 28MAR2019. Available at https://sassoftware.github.io/saspy/install.html

[25] SAS Institute. "Installation Note 46250: When you start SAS University Edition, an error might occur stating that VT-x or AMD-v is not available." *Samples and SAS Notes*. Date accessed: 28MAR2019. Available at http://support.sas.com/kb/46/250.html

[26] SAS Institute. "Introduction to the DS2 Language." *SAS 9.4 DS2 Language Reference, Sixth Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=ds2ref&docsetTarget=n1cievib65g7xbn1xvikyfgkfj6o.htm&docsetVersion=9.4&locale=en

[27] SAS Institute. "Introduction to the FedSQL Language." *SAS 9.4 FedSQL Language Reference, Third Edition*. Date accessed: 28MAR2019. Available at http://support.sas.com/documentation/cdl/en/fedsqlref/67364/HTML/default/viewer.htm#n1f3aii7i7qg27n1a6yvap9qf7pg.htm

[28] SAS Institute. "LUA Procedure." *SAS 9.4 Language Reference: Concepts, Sixth Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=proc&docsetTarget=n1w8nl91tml15dn1mw9p5l8oj6hy.htm&docsetVersion=9.4&locale=en

[29] SAS Institute. "Overview of SAS/IML Software." *SAS/IML 13.1 User's Guide*. Date accessed: 28MAR2019. Available at http://support.sas.com/documentation/cdl/en/imlug/66845/HTML/default/viewer.htm#imlug_imlstart_sect001.htm

[30] SAS Institute. "Pattern Matching Using Perl Regular Expressions (PRX)." *SAS 9.4 Functions and CALL Routines: Reference, Fifth Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=lefunctionsref&docsetTarget=n13as9vjfj7aokn1syvfyrpaj7z5.htm&docsetVersion=9.4&locale=en

[31] SAS Institute. "PRODUCT_STATUS Procedure." *Base SAS 9.4 Procedures Guide, Seventh Edition*. Date accessed: 28MAR2019. Available at https://documentation.sas.com/?docsetId=proc&docsetTarget=p167ky4zsoxrn2n1myz6iisao1wc.htm&docsetVersion=9.4&locale=en

[32] SAS Institute. "PROTO Procedure." *Base SAS 9.4 Procedures Guide, Seventh Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=proc&docsetTarget=n1wp77ftk3yeatn1jlkb2iojbttn.htm&docsetVersion=9.4&locale=en

[33] SAS Institute. "SAS_EM_PythonIntegration." *enlighten-integration*. Date accessed: 28MAR2019. Available at https://github.com/sassoftware/enlighten-integration

[34] SAS Institute. "SAS Kernel for Jupyter." *Open Source from SAS Software*. Date accessed: 28MAR2019. Available at https://github.com/sassoftware/sas_kernel

[35] SAS Institute. "SAS Kernel Magics." *Open Source from SAS Software*. Date accessed: 28MAR2019. Available at https://github.com/sassoftware/sas_kernel

[36] SAS Institute. "SAS Pipefitter." *Open Source from SAS Software*. Date accessed: 28MAR2019. Available at https://github.com/sassoftware/python-pipefitter

[37] SAS Institute. "SAS Scripting Wrapper for Analytics Transfer (SWAT)." *Open Source from SAS Software*. Date accessed: 28MAR2019. Available at https://github.com/sassoftware/python-swat

[38] SAS Institute. *SAS University Edition Download and Installation Guide*. Date accessed: 28MAR2019. Available at https://www.sas.com/en_us/software/university-edition/download-software.html

[39] SAS Institute. "Submit R Statements." *SAS/IML 13.1 User's Guide*. Date accessed: 28MAR2019. Available at http://support.sas.com/documentation/cdl/en/imlug/66845/HTML/default/viewer.htm#imlug_r_sect004.htm

[40] SAS Institute. "Using the Java Object." *SAS 9.4 Language Reference: Concepts, Sixth Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=lrcon&docsetTarget=n0swy2q7eouj2fn11g1o28q57v4u.htm&docsetVersion=9.4&locale=en

[41] SAS Institute. "What Is the SQL Procedure?" *SAS 9.4 SQL Procedure User's Guide, Fourth Edition*. Date accessed: 28MAR2019. Available at http://documentation.sas.com/?docsetId=sqlproc&docsetTarget=p1typbj1zqaum2n13o7mph0tdqsc.htm&docsetVersion=9.4&locale=en

[42] Seah, Aik Hoe. (2018) "A Method for Independent Program Validation utilising SAS, R and Python." *Proceedings of the PharmaSUG 2018 Conference*, Seattle, WA. Available at http://www.pharmasug.org/proceedings/2018/AD/PharmaSUG-2018-AD10.pdf

[43] Siegert, Stephen. (2018) "Test-Driven Data Science: Writing Unit Tests for SASPy Python Data Processes." *Proceedings of the SAS Global Forum 2018 Conference*, Denver, CO. Available at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2347-2018.pdf

[44] Slaughter, Susan. "What's your SAS interface?" *SAS Learning Post*. Date accessed: 28MAR2019. Available at https://blogs.sas.com/content/sastraining/2017/04/12/whats-your-sas-interface/

[45] Somers, James. (2018) "The Scientific Paper Is Obsolete: Here's what's next." *The Atlantic*. Date accessed: 28MAR2019. Available at https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsolete/556676/

[46] VanderPlas, Jake. (2016). A Whirlwind Tour of Python. O'Reilly Media: Sebastopol, CA. Available at https://jakevdp.github.io/WhirlwindTourOfPython/

[47] Van Rossum, Guido. (1996) "Foreword." *Programming Python, 1st Edition*. O'Reilly Media: Sebastopol, CA.

[48] Van Rossum, Guido. (1998) "Glue It All Together with Python." *Position paper for the OMG-DARPA-MCC Workshop on Compositional Software Architecture*. Date accessed: 28MAR2019. Available at https://www.python.org/doc/essays/omg-darpa-mcc-position/

[49] Winn, Thomas J., Jr. (2008) "Introduction to PROC TABULATE". *Proceedings of the SAS Global Forum 2008 Conference*, San Antonio, TX. Available at https://support.sas.com/resources/papers/proceedings/pdfs/sgf2008/171-2008.pdf

[50] Wyndham, Joe. (2018) "Fast, Flexible, Easy and Intuitive: How to Speed Up Your Pandas Projects." *Real Python*. Date accessed: 28MAR2019. Available at https://realpython.com/fast-flexible-pandas/

ACKNOWLEDGMENTS

The authors would like to thank Tasha Chapman and attendees of a Hands-on Workshop at the 2018 Western Users of SAS Software (WUSS) conference for their many helpful suggestions and feedback while creating this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the authors as follows:

Isaiah Lankham
Senior Research Analyst
U. of California Office of the President
1111 Franklin Street
Oakland, CA 94607
Phone: +1-510-987-9776
E-mail: Isaiah.Lankham@ucop.edu

Matthew Slaughter
Statistical Research Analyst
Kaiser Permanente Center for Health Research
3800 N Interstate Avenue
Portland, OR 97227
Phone: +1-503-335-2400
E-mail: Matthew.T.Slaughter@kpchr.org