Paper 156-2026

Enhancing Your SAS® Viya® Workflows with Python: Integrating Python's Open-Source Libraries with SAS® using PROC PYTHON

Ryan Paul Lafler, M.Sc., Premier Analytics Consulting, LLC
Miguel Angel Bravo, M.Sc., Premier Analytics Consulting, LLC

ABSTRACT

Data scientists, statistical programmers, machine learning engineers, and researchers are increasingly leveraging a growing number of open-source tools, libraries, and programming languages that can enhance and seamlessly integrate with their existing data workflows. One of these integrations, built into SAS® Viya®, is its pre-configured Python runtime integration, **PROC PYTHON**, that gives SAS programmers access to Python's open-source data science libraries for wrangling and modeling structured and unstructured data alongside the validated procedures provided in SAS. This paper demonstrates how to install and import external Python libraries into their SAS Viya sessions; generate Python scripts containing methods that can import, process, visualize, and analyze data; and execute those Python methods and scripts using SAS Viya's **PYTHON** procedure. By integrating the added functionalities of Python's libraries for data processing and modeling with SAS procedures, SAS programmers can enhance their existing data workflows with Python's open-source data solutions.

Keywords: *SAS® Viya®; PROC PYTHON; SAS-Python Integration; Open-Source Integration; Hybrid Analytics Workflows; Data Engineering; Pandas Library; SAS Dataset; Software Interoperability*

INTRODUCTION

Beginning in 2021, SAS® introduced the **PYTHON** procedure (**PROC PYTHON**) into their SAS® Viya® cloud analytics platform to integrate Python's growing community of open-source libraries and packages with their proprietary statistical programming language. This relatively new procedure integrates Python's objects, methods, libraries, and vibrant open-source community with the verified procedures developed by SAS. This integrated workflow offers advantages for both Python and SAS programmers, allowing them to co-exist in an integrated SAS Viya environment to build data processing pipelines and train supervised and unsupervised machine learning algorithms using Scikit-Learn, perform statistical modeling with **PROC GLM** and **PROC REG**, and handle a variety of structured, semi-structured, and unstructured data sources using Python libraries including Pandas, NumPy, OpenCV, and Xarray that come bundled with efficient, pre-built methods.

This application-oriented paper will demonstrate how to establish a connection between Python and SAS within SAS Viya using **PROC PYTHON** to execute Python scripts, install and import Python libraries within SAS Viya, define custom Python methods to help import and process data as an in-memory Pandas DataFrame, convert that Pandas DataFrame into a SAS dataset stored on disk, and investigate that SAS dataset exported from the Python code using base SAS procedures.

This paper was written to highlight the potential integration of Python and SAS programming within the SAS Viya ecosystem. By showing this integration, organizations with existing SAS Viya workflows can experiment with, adopt, and implement scalable data processing, analytical, and predictive systems using Python's well-supported community of libraries together with the validated procedures maintained by SAS.

1. Python Fundamentals

Among the most important open-source programming languages in 2024 is Python, which is an interpreted, object-oriented, and high-level programming language that features an active community of programmers and developers contributing libraries that are published on the Python Package Index (PyPi) and GitHub for anyone to download and use. As time advances, so do the languages used by organizations and businesses, which is why the integration of Python within SAS Viya offers SAS programmers new ways to extend the capabilities of SAS by incorporating open-source Python tools to assist in processing, visualizing, analyzing, and modeling data alongside the validated procedures created by SAS. Beginning with the basics, what exactly are the classes, objects, data structures, methods, and libraries in Python used for storing, accessing, manipulating, and handling data?

1.1 Python Data Structures, Objects, Classes, Methods, and Libraries

A class in Python is a template for creating objects, or unique instances, belonging to that class. A Python class acts as a blueprint for objects to share the same sets of attributes (options and methods) that can be customized and re-used when assigning unique objects. In other words, a class encapsulates the data and functions into a single, cohesive, and re-usable blueprint to customize objects from. Classes encourage the creation of complex and unique data structures through abstraction and code re-usability.

Python stores objects in different forms of mutable (editable) and immutable (permanent) data structures, including:

- **List** (indexed, mutable structure containing elements of the same or different types)
- **Tuple** (indexed, immutable structure of elements that cannot be changed once created)
- **Set** (unordered list of elements that cannot contain duplicate values)
- **Dictionary** (unordered, mutable JSON-like structure that relies on key → value mappings)

Methods in Python are typically defined within a class to carry-out tasks when creating objects. For example, a method can change the attributes of an object as defined by its class and return some type of value(s) to the user. Methods are defined by using the def keyword and are invoked using the dot notation (e.g., object.method()). Methods can modify/update/alter an object's state and perform operations using the object's parameters (data) passed to it by the programmer.

A Python library consists of a set of modules that contain re-usable, pre-defined, sharable, and customizable methods and classes developed and released in repositories like PyPi and GitHub by third-party authors. These modules typically include classes, methods, and additional dependency libraries that can be imported into Python programs to perform specialized (and often, optimized) tasks that reduce code duplication. All the methods of a library can be imported at once, or more efficiently, specific methods can be called only when needed. Some libraries, such as Scikit-Learn and TensorFlow, load more slowly when imported in their entirety as opposed to quicker-to-load libraries like NumPy and Pandas. All libraries must go through a one-time installation in the Python environment and must then be imported into the active Python session either in their entirety or by calling specific methods on an as-needed basis. Libraries facilitate code deployment by providing programmers with pre-built tools to perform specialized tasks with optimal efficiency and minimal code duplication (eliminates "re-creating the wheel").

NumPy extends the capabilities of Python to vectorize operations across entire columns (or rows) of data, rather than iterating operations sequentially one element at-a-time. Scikit-Learn integrates well with NumPy, SciPy, and Pandas to build data processing pipelines, train ML algorithms, and deploy them using minimal code and pre-defined classes in the form of data transformers and model estimators. These libraries greatly contribute to the scalability and efficiency of Python when building data workflows, also providing programmers with standardized frameworks of pre-built methods and documentation that teams can implement cohesively.

After covering these fundamental Python concepts, let's delve a little deeper into one of the most important Python libraries in the realm of data science and how it extends and enhances the capabilities of Python's more rudimentary data structures: the Pandas ecosystem.

1.2 Comparing the Pandas Library and its In-Memory DataFrame Objects to SAS Datasets

Pandas is an open-source Python library that provides high-performance data structures along with methods and tools for data cleaning, imputation, feature engineering, data analysis, and its own sets of feature types (called *dtypes*) that includes support for integer, float, string (*object*), categorical, datetime, and boolean types. Pandas is primarily used when working with tabular data (i.e., flat file formats such as CSV, JSON, spreadsheets) and includes methods for scraping data from the web and working with unstructured text-based data sources.

Pandas assists with tasks including data imports, data cleaning, filtering, subsetting, aggregating, summarizing, transforming, time series management (using its vectorized datetime functions), and data exports using its ecosystem of well-defined methods. The library offers two main structures: the DataFrame (tabular data comprised of rows and columns) and a Series (a vector of data stored as either a row *or* column inside a DataFrame). A Pandas DataFrame can be created from a Python dictionary containing lists of elements where the keys of the dictionary denote the DataFrame's column names (features), and the values (stored in a list) for each key produce the values for that specific column.

How are Pandas DataFrames and SAS datasets alike? Both share similarities in how they handle and organize data in a 2-dimensional tabular format, particularly for data analysis, querying, and manipulation tasks. Pandas DataFrames and SAS datasets both support:

- Mixed feature types including numeric, string, categorical, and datetime column types
- Encoding and representation of missing values (Pandas uses '*NaN*' while SAS employs '**.**' to denote missing values)
- Merging, joining, stacking, and concatenation between different datasets
- Generating statistical summaries that recognize and account for missing values

Although Pandas DataFrames and SAS datasets share many similarities in their formats and structures, there are, however, several key differences with how data are retrieved, indexed, stored, and the methods in which data can be manipulated and transformed. These differences include:

- Pandas DataFrames must be loaded entirely in Python's memory (RAM), making methods such as data chunking and lazy evaluation necessary with additional libraries like Dask and Polars. SAS datasets are stored on disk, allowing SAS to naturally (out-of-the-box) process big data concurrently without running into memory constraints or limitations.

- Methods for processing, sorting, filtering, transforming, and engineering new features are built into DataFrame objects. These operations are performed entirely in Python's memory. Python programmers can even define, vectorize, and map their own custom functions to DataFrames. SAS datasets require the use of pre-defined procedures (PROCs) to perform similar functions.

- Pandas DataFrames support automatic and custom (multi- and hierarchical) indexing to denote rows and columns, while SAS datasets require that indices be *explicitly* created on one or more columns. Since SAS datasets read data from the disk, as opposed to Python DataFrames where the entire index is stored in memory, filtering by indices on a SAS dataset for moderate-sized data is slower than its Python counterpart.

Therefore, when using **`PROC PYTHON`** in SAS Viya and importing data using the Pandas library, the entire dataset will be imported into the Python session's memory (stored for immediate access with RAM). This stands in contrast to the **`DATA`** step in base SAS where the data are read from the disk and chunks of the data are loaded into SAS as needed (constant I/O for read/write operations into SAS). ***Figure 1*** shows a comparison of how the Pandas DataFrame and SAS dataset visually appear, respectively, using the same tabular dataset source. Notice how Python automatically indexes the first observation starting at 0 and then increments by 1 from its 0-based index (this is a significant and noteworthy difference from SAS).

	PatientID	Age	Gender	Ethnicity	SocioeconomicStatus	EducationLevel	BMI	Smoking	AlcoholConsumption	PhysicalActivity	...	Itching	Qualit
0	1	71	0	0	0	2	31.069414	1	5.128112	1.676220	...	7.556302	
1	2	34	0	0	1	3	29.692119	1	18.609552	8.377574	...	6.836766	
2	3	80	1	1	0	1	37.394822	1	11.882429	9.607401	...	2.144722	
3	4	40	0	2	0	1	31.329680	0	16.020165	0.408871	...	7.077188	
4	5	43	0	1	1	2	23.726311	0	7.944146	0.780319	...	3.553118	

KIDNEY_SAS | Table rows: 1659 | Columns: 54 of 54 | Rows 1 to 200

Enter expression

	PatientID	Age	Gender	Ethnicity	SocioeconomicStatus	EducationLevel	E
1	1	71	0	0	0	2	31.0694
2	2	34	0	0	1	3	29.6921
3	3	80	1	1	0	1	37.3948
4	4	40	0	2	0	1	31.3296
5	5	43	0	1	1	2	23.7263

Figure 1. Pandas DataFrame object (top) and the equivalent SAS dataset (bottom) showing the first-5 observations. Notice how Python implements automatic indexing starting at 0.

1.3 Building a Custom Python Method to Import External CSV Data Sources

Leveraging pre-built methods from the Pandas library, let's define our own Python method that can accomplish the following data import tasks:

- Import any delimited file type (i.e., comma, tab, colon) into the Python session as a Pandas DataFrame
- Allow programmers to specify the input file name, exported SAS dataset name, and type of delimiter
- Elements in the first row of the delimited file are used as column header names
- Convert the Pandas DataFrame from an in-memory dataset to a SAS dataset stored on-disk in default **`WORK`** folder
- Allow programmer to determine if data is temporarily saved or permanently saved on-disk using a **`LIBNAME`**

This custom method, defined in **Example 2,** is saved inside of a separate Python script file (with the **`.py`** file extension) as **`import_data.py`**. Python script files can contain as many user-defined methods, classes, and objects that can be imported and used directly inside of **`PROC PYTHON`** through the **`INFILE`** option and pointing it to the script file's location in SAS Drive. For simplicity, the authors recommend saving the Python script file, SAS program file, and CSV dataset to the same working folder inside of SAS Viya's SAS Drive.

Example 1. Using an external Python script (.py) file to define a method for importing CSV datasets: `import_data.py`

```
def import_data(self, file_name, export_name, delim=',', libname='/') :
        try :
                # Import using the Pandas library
                df = pd.read_csv(
                        file_name ,
                        delimiter = delim ,
                        header = 'infer'
                )

                # Export pathway
                export_path = f'{libname}.{export_name}'
                # Transform into a SAS Dataset
                SAS.df2sd(df, export_path)

        except :
                raise Exception(
                        f'There was an issue importing {file_name}'
                )
```

The Python script file shown in **Example 1** defines a custom method called **`import_data()`** that imports any delimited data source as a Pandas DataFrame and converts it to a temporary SAS dataset. When passing this script into **PROC PYTHON** and running it, a new Python runtime *within* the current SAS Viya SAS session will be launched, and the CSV dataset is then imported as an in-memory Pandas DataFrame and then converted and exported as a SAS dataset to the default **WORK** directory.

The **`import_data()`** method shown in **Example 1** also incorporates a **`try-except`** statement to assist in debugging. Should an error occur anywhere within the **`try`** block, then the **`except`** block executes and the error message is returned. If the **`try`** block executes successfully, then the error message in the **`except`** block is not returned.

By default, the **LIBNAME** option is set to store the SAS dataset in on-disk temporary storage, typically called the **WORK** directory. Any files stored inside of this temporary storage are deleted when the SAS Viya session is terminated. Programmers can persist the resulting SAS dataset to permanent storage in the SAS Drive by changing the **LIBNAME** option to a permanent folder pathway.

2. Connecting to PROC PYTHON in SAS Viya

With the Python script file containing the import method saved and exported, let's focus on initializing a Python session within SAS Viya using **PROC PYTHON**. Let's first investigate the version of Python that has been installed in **PROC PYTHON**. Then let's examine which open-source libraries (and their versions) are installed in the SAS Viya Python environment.

Traditionally, the interaction between SAS and Python happens through the **SASPy** library. **PROC PYTHON** simplifies this process by coming bundled with a pre-configured, *internal*, and updated version of Python that does not need to be externally connected to the SAS environment, since **PROC PYTHON** is already running *within* the SAS Viya environment. This simplifies data workflows and session management between Python and SAS Viya but limits the programmer to using only the pre-installed Python environment and installed libraries within **PROC PYTHON**. More flexibility is allowed by fully using the **SASPy** library to connect with external Python runtimes, also allowing users of SAS 9.4 *and* SAS Viya to connect their own customized Python environments to their SAS sessions, albeit with more complexity to manage the connection between sessions and properly setup.

2.1 Checking the Installed Python Version in SAS Viya

Knowing the Python version installed within **PROC PYTHON** is important when managing library dependencies, ensuring compatibility between packages, and avoiding version conflicts that may affect analytical workflows. This can be achieved by importing Python's *sys* library, which is installed alongside Python natively, and allows the programmer to interact directly with the Python runtime environment installed within **PROC PYTHON**.

This information can be obtained by importing Python's **`sys`** library, which is installed alongside Python natively and allows the programmer to interact directly with the Python runtime environment available within SAS Viya.

The **PROC PYTHON** code shown in **Example 2** includes a multi-line Python code block that prints the currently installed Python version used in the procedure. The version of Python available in **PROC PYTHON** may change over time as SAS Viya environments are updated, so verifying the installed version helps ensure compatibility with Python libraries and scripts used within SAS workflows.

Example 2. Checking the Python Version Installed in SAS Viya's PROC PYTHON

```
/* Check and verify Python Installation in SAS Viya */
PROC PYTHON ;
        SUBMIT ;

import sys
print(f'Installed Python Version: {sys.version}')

        ENDSUBMIT ;
RUN ;
```

```
85   PROC PYTHON ;
86   SUBMIT
NOTE: Python initialized.
Python 3.9.18 (main, Jun 25 2024, 16:00:09)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-20)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
86 !         ;
87
88   import sys
89
90   print(f'Installed Python Version: {sys.version}')
91
92   ENDSUBMIT ;
93   RUN ;
>>>
Installed Python Version: 3.9.18 (main, Jun 25 2024, 16:00:09)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-20)]
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time           0.68 seconds
      cpu time            0.02 seconds
```

(Example 2) SAS Log and Python Output

The SAS log produced from running the **PROC PYTHON** code in **Example 2** shows the installed Python version as 3.9.18 running on a Red Hat Linux operating system. When executing a **PROC PYTHON** block for the first time in a session, the log also displays information about the Python runtime and the operating system environment in which it is running. This information is useful when troubleshooting library installations, dependency issues, or environment-specific behavior when integrating Python workflows within SAS Viya.

2.2 Verifying Installed Python Libraries in the PROC PYTHON Session

After identifying the Python runtime version available within **PROC PYTHON**, the next step is to determine which Python libraries are installed in the SAS Viya environment. Since the embedded Python environment is periodically updated, verifying installed libraries and their versions is important for managing dependencies, ensuring compatibility between scripts, and supporting reproducible analytical workflows.

This can be accomplished using the **sys** and **subprocess** libraries, both of which are installed with Python and allow the programmer to issue pip terminal commands that interact with the Python runtime environment. The code shown in **Example 3** uses the **pip freeze** command to retrieve a listing of all installed packages (and versions) and prints the results to the SAS log.

Example 3. Checking Library Installations in PROC PYTHON Runtime

```
/* Check installation of Python libraries in SAS Viya */
PROC PYTHON ;
    SUBMIT ;

# Import subprocess to run pip commands by accessing the Python terminal in SAS Viya
import sys
import subprocess
```

```
# Run the pip freeze command to list all installed packages
installed_packages = subprocess.check_output([sys.executable, '-m', 'pip', 'freeze'])

# Decode and print the result
print(installed_packages.decode("utf-8"))

    ENDSUBMIT ;
RUN ;
```

```
96   /* Check installation of Python libraries in system */
97   PROC PYTHON;
98        SUBMIT
NOTE: Resuming Python state from previous PROC PYTHON invocation.
98 !           ;
99
100  # Import subprocess to run pip commands
101  import subprocess
102
103  # Run the pip freeze command to list all installed packages
104  installed_packages = subprocess.check_output([sys.executable, '-m', 'pip', 'freeze'])
105
106  # Decode and print the result
107  print(installed_packages.decode("utf-8"))
108
109       ENDSUBMIT;
110  RUN;
>>>
absl-py==2.1.0
aiohttp==3.9.5
aiosignal==1.3.1
annotated-types==0.7.0
asttokens==2.4.1
astunparse==1.6.3
async-timeout==4.0.3
attrs==23.2.0
blinker==1.8.2
```

(Example 3) SAS Log and Python Output

The SAS log from **Example 3** shows a subset of the Python libraries installed within the **PROC PYTHON** environment. These libraries include tools for processing structured and unstructured data (e.g., Pandas, GeoPandas, NumPy, SciPy, Pillow, NLTK, Regex), accessing data from cloud storage and web services (e.g., Filesystem Spec, AIOHTTP), data visualization (e.g., Matplotlib, Plotly, Seaborn), statistical modeling and machine learning (e.g., Statsmodels, Scikit-Learn, XGBoost), and deep learning frameworks (e.g., TensorFlow, PyTorch, Keras, NVIDIA CUDA toolkit). The availability of these libraries allows PROC PYTHON to support a wide range of data processing, machine learning, and analytical workflows within the SAS Viya environment.

After verifying the installed Python libraries available within **PROC PYTHON**, the next step is to begin incorporating Python data processing workflows within SAS Viya by importing external data sources and converting them into SAS datasets.

3. Incorporating Data Processing Workflows in SAS Viya with PROC PYTHON

With the Python script file containing the custom import method saved to the SAS Drive and the **PROC PYTHON** environment providing access to the Pandas library, we can now use **PROC PYTHON** to import a CSV dataset from the SAS Drive using Pandas and export the resulting Pandas DataFrame as a SAS dataset stored temporarily in the **WORK** directory.

After examining the Python runtime environment available within **PROC PYTHON** and the installed libraries, we can now construct a practical workflow that integrates Python data processing with SAS dataset creation and storage.

3.1 Declaring File Reference and Macro for use between SAS and PROC PYTHON

The dataset and Python script file must first be uploaded into the SAS Drive. The next step is to define file pathways to both the Python script file and the CSV dataset so that they can be accessed within both the SAS session and the PROC PYTHON environment. The SAS code in **Example 4** achieves the following:

- Creates a file reference called **SCRIPT** for the Python script file **import_data.py** stored on disk (SAS Drive) using the **FILENAME** statement
- Declares a macro variable called **REFDATA** that stores the CSV dataset pathway for use within **PROC PYTHON**

In the SAS code in **Example 4**, two items are defined: a file reference for the Python script file and a macro variable containing the pathway to the CSV dataset. By defining a file reference with the **FILENAME** statement, programmers can use the shorthand reference name **SCRIPT** instead of its full pathway when importing the Python script into **PROC PYTHON**. The **REFDATA** macro variable allows the pathway to the CSV dataset to be defined in SAS and passed to the Python session as a string input. Code in **Example 4** shows how to implement this in SAS Viya.

Example 4. Defining a Reference to the Python Script File on Disk (SAS Drive) and Macro Variable Containing File Pathway

```
/* Python script file input */
FILENAME SCRIPT DISK '/export/viya/homes/{user-email}/casuser/import_data.py' ;

/* Define macro variable for dataset path */
%LET REFDATA = /export/viya/homes/{user-email}/casuser/Chronic_Kidney_Disease_data.csv ;
```

3.2 Importing the CSV File with PROC PYTHON as a Pandas DataFrame

A Python session within SAS Viya is initialized by calling **PROC PYTHON** and specifying the Python script file using the **INFILE** option when executing external **.py** files. In this example, the Python code from **Example 1** was saved to an external Python script file, using the **.py** extension, containing a custom method for importing delimited data sources using the Pandas library and then exporting that Pandas DataFrame as a SAS dataset that can be accessed by any of SAS Viya's procedures.

When executing a Python code block inside of **PROC PYTHON**, programmers must specify the **SUBMIT** and **ENDSUBMIT** statements to mark the beginning and end of the executed Python code. The Python code within the **SUBMIT** and **ENDSUBMIT** block is executed sequentially, line by line, until either successful completion or termination due to an error.

The **PROC PYTHON** code shown in **Example 5** can be summarized by the following steps:

1. Imports the Python script file using the **FILENAME** reference called **SCRIPT** from **Example 4**
2. Imports the pre-installed Pandas library into the active **PROC PYTHON** environment
3. **SAS.symget('REFDATA')** retrieves the file pathway stored in the **REFDATA** macro variable and assigns it to a Python string object called **filename**
4. **import_data(...)** method imports the CSV dataset as an in-memory DataFrame inside of Python environment
5. **import_data(...)** finishes by exporting the in-memory DataFrame to a SAS dataset stored on disk in **WORK** library

Example 5. Importing a CSV Dataset using PROC PYTHON and Exporting as a SAS Dataset

```
/* Submit Python code into PROC PYTHON */
PROC PYTHON INFILE=SCRIPT ;
        SUBMIT ;

# Python library imports
import pandas as pd

# Specify filename pathway
filename = SAS.symget('REFDATA')

# Retrieve method from Python script file to load the data
import_data(
        file_name = filename ,
        export_name = 'KIDNEY_SAS'
)


        ENDSUBMIT ;
RUN ;
```

```
>>>
135  proc printto
135! log='/opt/sas/viya/config/var/tmp/compsrv/default/edcd2a8d-f34d-4148-a968-577345d10cf0/SAS_workDB4200000202_sas-compute-server-
135! b5e7b800-42b2-4932-8634-13ba62305b81-6956/sas2py.log';run;filename sock socket 'localhost:53021' recfm=V termstr=LF
NOTE: PROCEDURE PRINTTO used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds

>>>
/export/viya/homes/rplafler@sdsu.edu/casuser/Papers/PROC-PYTHON/Chronic_Kidney_Disease_data.csv
>>>
NOTE: PROCEDURE PYTHON used (Total process time):
      real time           0.17 seconds
      cpu time            0.03 seconds
```

(Example 5) SAS Log

The code in **Example 5** can also retrieve and edit macro variables defined in the SAS session inside **PROC PYTHON**, which is completed using the SASPy method **`SAS.symget('REFDATA')`**, containing a macro variable string defined by the **%LET** statement called **REFDATA** that contains the file pathway to the CSV file. Without requiring additional libraries, **PROC PYTHON** allows direct interaction between SAS macro variables and Python objects, which in this case, takes the macro variable **REFDATA** and stores it as a Python string called **`filename`**.

Once this code block is executed, a new SAS dataset (**`.sas7bdat`**) file called **KIDNEY_SAS** is created and stored (temporarily) inside of the **WORK** library and can now be accessed by any of the descriptive, visualization, statistical, and modeling procedures in SAS.

4. Persisting the Created SAS Dataset to Permanent Storage

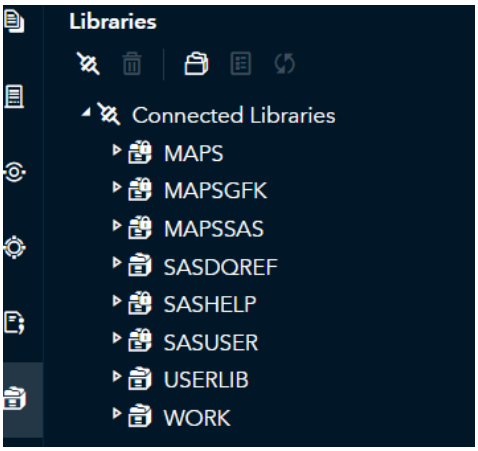
Currently, the resulting SAS dataset is saved in temporary storage in the **WORK** library. By setting a **LIBNAME** in the SAS code that points to a persistent folder in the SAS Drive, programmers can utilize the SAS dataset across SAS Viya sessions without having to import and process the original CSV data source in Pandas.

4.1 Defining a Persistent Storage Directory with LIBNAME and LIBREF in SAS Viya

The **LIBNAME** statement allows the programmer to define a persistent directory that can be invoked using an 8-character (max) shorthand reference within SAS for convenient dataset storage. The **LIBREF** contains the reference to the dataset folder pathway while the **LIBNAME** is the shorthand library name. The **LIBNAME** statement shown in **Example 6** defines a library named **USERLIB** that is stored in this designated folder location: **`/export/viya/homes/{user-email}/casuser/Data`**. The /Data folder was created to store and persist SAS datasets between SAS Viya sessions.

Example 6. Defining a LIBNAME and LIBREF for Persistent SAS Drive Data Storage

```
/* Create a persistent folder containing exported SAS Datasets */
LIBNAME USERLIB '/export/viya/homes/{user-email}/casuser/Data' ;
```



(Example 6) Library Output Showing USERLIB Library

Located inside of SAS Studio's libraries tab, the output from **Example 6** creates the **USERLIB** library that is added to the existing list of default (locked) libraries and the temporary **WORK** library.

4.2 Using the LIBNAME in PROC PYTHON to Export the Pandas DataFrame as a SAS Dataset to Persistent Storage (USERLIB)

After setting up the persistent library connected with the location in SAS Drive, let's amend the **PROC PYTHON** code to account for this by setting the **LIBNAME** option in the **`import_data()`** Python method to point to the **USERLIB** library and persist the exported SAS dataset to its designated reference folder. The SAS code in **Example 7** shows how to do this.

Example 7. Using PROC PYTHON to Import the CSV File, Export, and Persist the SAS Dataset Using LIBNAME

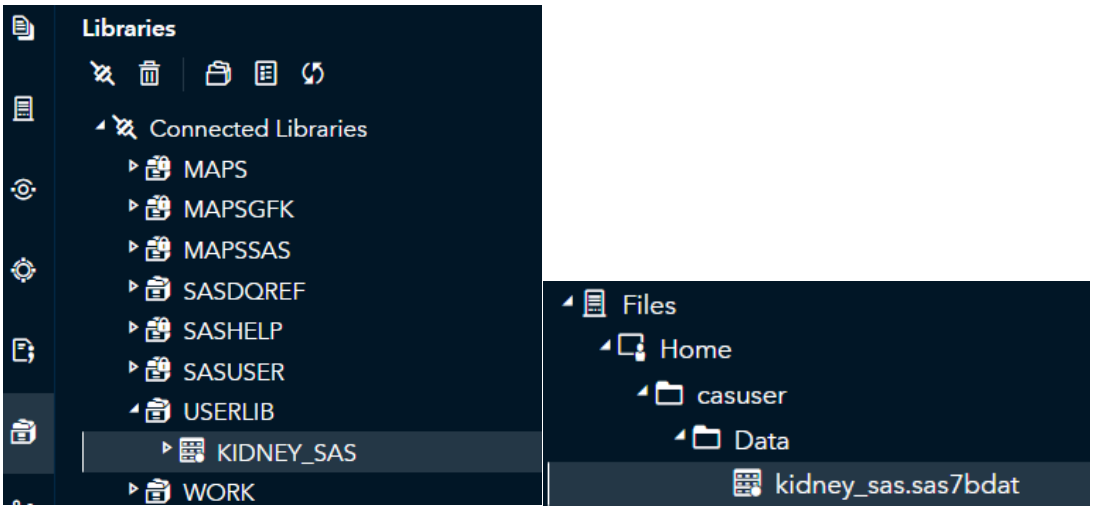
```
/* Submit Python code into PROC PYTHON */
PROC PYTHON INFILE=SCRIPT ;
        SUBMIT ;

# Python library imports
import pandas as pd

# Specify filename pathway
filename = SAS.symget('REFDATA')

# Retrieve method from Python script file to load the data
import_data(
        file_name = filename ,
        export_name = 'KIDNEY_SAS' ,
        libname = 'USERLIB'  # Set the LIBNAME to the `USERLIB` library
)


        ENDSUBMIT ;
RUN ;
```



(Example 7) Output Showing the SAS Dataset Exported and Stored in USERLIB

As shown in the SAS output for **Example 7**, the original CSV dataset is successfully exported as a SAS dataset file (**`.sas7bdat`**) that persists between SAS Viya sessions. In summary, the original CSV dataset was imported into the **PROC PYTHON** runtime as an in-memory Pandas DataFrame and then converted into a SAS dataset that was exported and saved to the **LIBREF** location specified in the **LIBNAME** statement. This process showcases a framework for integrating the data wrangling capabilities of Python within the SAS environment to generate SAS datasets from raw data sources that are now accessible by any of the procedures in SAS workflows.

5. Examining the Metadata of the Newly Created SAS Dataset

Finally, let's verify the integrity of the resulting SAS dataset to ensure that the data import, conversion, and export processes performed in **PROC PYTHON** successfully transformed the raw CSV file. Let's use **PROC CONTENTS** to generate and investigate the metadata for this resulting SAS dataset.

The SAS code in **Example 7** invokes **PROC CONTENTS** on the SAS dataset exported and saved to the **USERLIB** library.

Example 7. PROC CONTENTS on the Persisted SAS Dataset

```
/* Check the contents of the created SAS Dataset */
PROC CONTENTS DATA=USERLIB.KIDNEY_SAS ;
        TITLE 'Metadata for the SAS Dataset' ;
RUN ;
```

Metadata for the SAS Dataset

The CONTENTS Procedure

Data Set Name	USERLIB.KIDNEY_SAS	**Observations**	1659
Member Type	DATA	**Variables**	54
Engine	V9	**Indexes**	0
Created	10/21/2024 20:10:27	**Observation Length**	440
Last Modified	10/21/2024 20:10:27	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64, LINUX_POWER_64		
Encoding	utf-8 Unicode (UTF-8)		

(Example 7) PROC CONTENTS Output for Saved SAS Dataset: USERLIB.KIDNEY_SAS

The **PROC CONTENTS** output shows there are a total of 1,659 observations and 54 features within this dataset. Each observation accounts for approximately 440 bytes of data on disk. Therefore, it's safe to assume that the integration of Python within SAS was successful.

6. Discussion and Industry Applications

The integration of Python within SAS Viya using **PROC PYTHON** supports a hybrid analytics workflow model that allows organizations to incorporate open-source tools while maintaining existing SAS-based workflows, validation processes, and production analytics environments. Rather than replacing SAS workflows, **PROC PYTHON** enables Python to be used alongside SAS as a complementary tool for data ingestion, preprocessing, visualization, machine learning, and automation tasks, while SAS continues to be used for validated statistical procedures, reporting, and regulated analytics workflows.

This hybrid workflow approach is particularly valuable in regulated industries such as pharmaceuticals, clinical research, finance, insurance, and government analytics environments, where SAS has long been used for validated statistical analysis, regulatory reporting, and controlled production workflows. In these environments, organizations often want to adopt open-source tools such as Python for data engineering, machine learning, and automation, but must still maintain validated and governed analytics workflows within SAS environments. **PROC PYTHON** provides a practical framework for integrating these open-source tools into SAS Viya workflows without disrupting existing validation, governance, and reporting processes.

In practice, Python can be used within **PROC PYTHON** as a data processing and machine learning layer that imports external data sources, performs feature engineering, trains machine learning and deep learning models, processes unstructured data, or automates data ingestion tasks. The resulting datasets can then be exported as SAS datasets and integrated into existing SAS workflows for statistical analysis, reporting, and production analytics processes. This workflow architecture allows organizations to leverage Python's open-source ecosystem while continuing to rely on SAS for validated analytics workflows and enterprise reporting systems.

Another important consideration when using **PROC PYTHON** is the difference between Python's in-memory data processing model and SAS's disk-based data processing model. Pandas DataFrame objects operate entirely in memory, which can provide faster processing for moderate-sized datasets but may require chunking or distributed processing libraries for very large datasets. SAS datasets, on the other hand, are stored on disk and processed in chunks, making SAS well-suited for large-scale data processing workflows. Understanding how these two processing models differ allows programmers to design hybrid workflows that leverage Python and SAS where each tool is most effective.

Overall, **PROC PYTHON** enables organizations to build integrated SAS and Python workflows that combine the flexibility of open-source tools with the stability, governance, and validated procedures provided by SAS. This hybrid integration model allows organizations to extend existing SAS environments with open-source capabilities for data engineering, machine learning,

automation, and advanced analytics while maintaining established SAS infrastructure and validated analytical processes. As a result, integrated SAS and Python environments are becoming increasingly common in enterprise and regulated analytics settings where organizations must balance innovation with governance, reproducibility, and regulatory compliance.

CONCLUSION

PROC PYTHON provides a powerful framework for integrating Python's open-source ecosystem directly within SAS Viya workflows, allowing programmers to combine Python's data processing, visualization, and machine learning capabilities with the validated and production-ready procedures available in SAS. This integration enables organizations to extend existing SAS workflows without replacing established SAS infrastructure, creating hybrid analytics workflows that leverage the strengths of both environments.

The workflow demonstrated in this paper illustrates how Python can be used to import and process external data sources using Pandas, convert those in-memory DataFrame objects into SAS datasets, and persist those datasets for use across SAS Viya sessions. This approach allows Python to serve as a flexible data ingestion and preprocessing layer while SAS continues to provide robust statistical procedures, reporting, and production analytics capabilities.

As organizations increasingly adopt open-source tools alongside established enterprise analytics platforms, integrated workflows between SAS and Python will become increasingly important for building scalable, reproducible, and production-ready analytical systems. **PROC PYTHON** provides a practical and accessible framework for SAS programmers to incorporate Python into their existing workflows while maintaining the governance, validation, and reliability that SAS environments provide.

REFERENCES

Box, J. (2023). *Running Python code inside a SAS® program*. In *Proceedings of PharmaSUG 2023 Conference* (Paper QT-165). Retrieved from https://pharmasug.org/proceedings/2023/QT/PharmaSUG-2023-QT-165.pdf

SAS Institute. (2024, October 17). *How PROC PYTHON Works*. SAS® Help Center. https://documentation.sas.com/doc/en/pgmsascdc/v_056/proc/p1m1pc8yl1crtkn165q2d4njnip1.htm

ACKNOWLEDGEMENTS

The authors would like to acknowledge the Western Users of SAS® Software (WUSS) 2026 Conference Committee, including the Academic Chair, Operations Chair, and Section Chair, for their acceptance of this submission and for the opportunity to present and publish this work at WUSS 2026.

AUTHORS' BIOGRAPHIES

Ryan Paul Lafler, M.Sc., is the Founder, CEO, and Lead Consultant of Premier Analytics Consulting, LLC, a California-certified small business based in San Diego specializing in localized AI and machine learning systems, data engineering, enterprise GIS, clinical and statistical analysis, and full-stack analytics platform development. Through project-based assignments and consulting roles, Ryan provides advisory, development, and training services for custom AI/ML systems, full-stack platforms, data engineering infrastructure, GIS solutions, and statistical and clinical workflows for businesses, public-sector agencies, and research institutions. Leading a specialized consulting team, Ryan's cross-industry expertise includes systems architecture, integration, and enterprise platform development using Python, R, SAS®, SQL/NoSQL, and modern JavaScript frameworks. He also serves as an Adjunct Professor in the Big Data Analytics Graduate Program and the Department of Mathematics and Statistics at San Diego State University. He earned his Master of Science in Big Data Analytics following the publication of his thesis, and his Bachelor of Science in Statistics with a Minor in Quantitative Economics, from San Diego State University.

~ ~ ~ ~ ~ ~ ~

Miguel Angel Bravo, M.Sc. is a Consultant and Data Scientist for Premier Analytics Consulting, LLC, where he develops applied machine learning systems, AI integrations, big data pipelines, and data-driven full-stack platforms for enterprise analytics. His work spans production-ready AI/ML workflows, containerized analytics platforms, and data engineering with Python, Docker, AWS/GCS/Azure, and MLOps practices. Miguel holds a Master of Science in Big Data Analytics from San Diego State University and a Bachelor of Science in Electronics, Robotics, and Mechatronics Engineering from the University of Málaga, with research experience in environmental modeling, geospatial analytics, and AI-driven decision support systems.

CONTACT INFORMATION

Comments, suggestions, and/or any questions are encouraged and may be sent to:

Ryan Paul Lafler, M.Sc.
Founder, CEO, and Lead Consultant
Premier Analytics Consulting, LLC
Email: rplafler@premier-analytics.com
Website: www.Premier-Analytics.com
LinkedIn: www.linkedin.com/in/RyanPaulLafler

~ ~ ~ ~ ~ ~ ~

Miguel Angel Bravo, M.Sc.
Consultant and Data Scientist
Premier Analytics Consulting, LLC
Email: mabravo@premier-analytics.com

COMPANY INFORMATION

Premier Analytics Consulting, LLC is a California Certified Small Business based in San Diego specializing in the design, development, and integration of technical systems for organizations operating across complex and large data environments. Our firm specializes in AI and machine learning systems, data engineering infrastructure, clinical and statistical analysis, enterprise GIS solutions, open-source modernization efforts, and full-stack platform development to provide comprehensive support to each client. From advisory and evaluation through implementation, deployment, documentation, and training, our services align to each client's analytical, data, infrastructure, security, and operational needs. Premier Analytics Consulting works with American and international businesses, public-sector agencies, and research institutions through remote and hybrid contracts, subcontracts, consulting engagements, technical partnerships, and training services.

➤ ***Learn more about our services, products, and engagement structures:*** **www.Premier-Analytics.com**

TRADEMARK CITATIONS

Premier Analytics Consulting, LLC, Premier Training, Premier AI, and their associated logos are trademarks of Premier Analytics Consulting, LLC. SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. All other brand and product names are trademarks of their respective owners.