Paper 3415-2019

SAS® and Open Source: Two Integrated Worlds

Jesse Luebbert and Radhikha Myneni, SAS Institute Inc.

ABSTRACT

Data scientists use a variety of tools, both commercial and open-source, to achieve key goals for their organization. For enterprise applications of analytics and artificial intelligence, it is crucial that teams can collaborate no matter which tools they are using. SAS® software provides a platform on which all users in the enterprise can create intelligence from data and operationalize the results easily. Data scientists and developers whose core programming competence is in languages such as Python and R can efficiently use SAS through a variety of APIs to increase productivity and improve time-to-value. This paper describes and demonstrates a variety of best-practice use cases to show how SAS software provides integration with open-source tools to support end-to-end analytical workflows.

INTRODUCTION

SAS has a long history of providing high-quality statistical, data mining, and machine learning software for various industries. SAS offers solutions to build credit scorecards, detect fraud, assess risk, or provide recommendations that automate and streamline decision-making processes. In recent years, many data scientists have used SAS software, Python, R, and other open-source or vendor-specific tools to mix and match various tasks of the analytical life cycle. To enable these combinations, SAS provides Python and R integration with multiple releases of SAS 9, and continues to extend these capabilities with SAS® Viya®, the cloud-enabled, in-memory, distributed analytics engine that makes the SAS Platform more scalable, fault-tolerant, and open. The word "open" signifies the fact that the power of SAS to build and deploy analytics can be accessed via many programming languages—not just SAS, but also Python, R, Lua, Java, or RESTful APIs. This integration enables analytical teams with varied backgrounds and experiences to come together and solve complex problems in new ways.

Integrating SAS and open-source technologies is often advantageous in two main scenarios:

- Programmatically accessing the SAS Platform using open-source software
- Bringing open-source models into the SAS Platform for side-by-side comparison

Each of these topics is discussed in detail with examples in the following sections.

SAS TO OPEN-SOURCE LANGUAGES AND INTERFACES

This section starts with an example of being able to access and execute SAS analytics programmatically from open-source languages. For consistency, the primary focus is on calling SAS from Python, a popular general-purpose scripting language, via APIs. SAS provides three foundational open-source packages for doing this, all available on GitHub: SASPy, SWAT,[1] and ESPPy. The SASPy package interfaces with SAS 9.4, SWAT with SAS Viya, and ESPPy with SAS® Event Stream Processing. Three higher-level open-source packages are also available: Pipefitter, SASOptPy, and DLPy. All these packages are open-source, and contributions from the community are welcome. Figure 1 shows a visual representation of the packages with their dependencies.

[1] There is also a SWAT package for R.

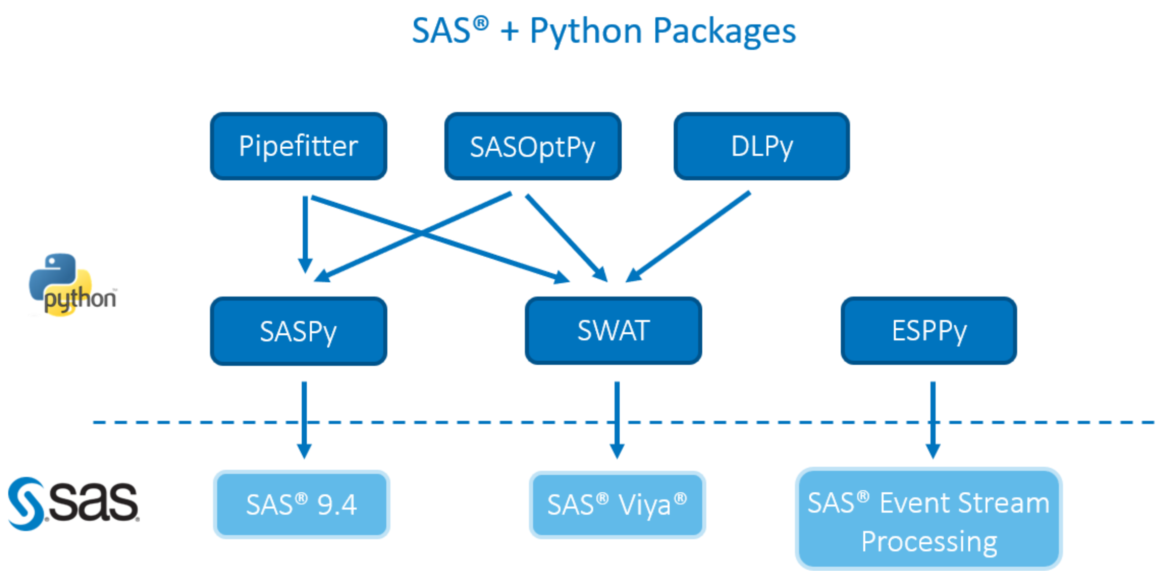

Figure 1. SAS and Python Packages

SASPY

SASPy is a Python module that interfaces between Python 3.x or later and SAS 9.4 or later and between Python 3.x or later and all releases of SAS Viya. At a minimum, you can use SASPy to run existing SAS code, procedures, DATA steps, and so on from a Python method called submit. Figure 2 shows an example of using SASPy to run the PRINT procedure in SAS from Python.

Run SAS code directly in Python! Get the output (LST) and log

```
In [11]: ll = sas.submit('proc print data = sashelp.cars (obs = 5); run;')
```

```
In [12]: HTML(ll['LST'])
```

Out[12]:

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197

```
In [13]: print(ll['LOG'])
```

```
18                                                        The SAS System                          14:32 Wednesday, February 28,
2018

153        ods listing close;ods html5 (id=saspy_internal) file=_tomods1 options(bitmap_mode='inline') device=svg; ods graphics
on /
153      !  outputfmt=png;
NOTE: Writing HTML5(SASPY_INTERNAL) Body file: _TOMODS1
154
155        proc print data = sashelp.cars (obs = 5); run;

NOTE: There were 5 observations read from the data set SASHELP.CARS.
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.02 seconds
      cpu time            0.02 seconds


156
157        ods html5 (id=saspy_internal) close;ods listing;
158
```

Figure 2. PROC PRINT Example That Uses SASPy

Another benefit is the easy transfer of data from Python to SAS and vice versa. This can be very beneficial when you want to use elements of multiple languages to mix and match workflows.

You can transfer data between SAS data sets and pandas data frames.

```
In [18]: import pandas
```

```
In [19]: car_df = cars.to_df()
```

```
In [20]: type(car_df)
Out[20]: pandas.core.frame.DataFrame
```

```
In [21]: car_df.head()
```

Out[21]:

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
0	Acura	MDX	SUV	Asia	All	36945	33337	3.5	6.0	265	17	23	4451	106	189
1	Acura	RSX Type S 2dr	Sedan	Asia	Front	23820	21761	2.0	4.0	200	24	31	2778	101	172
2	Acura	TSX 4dr	Sedan	Asia	Front	26990	24647	2.4	4.0	200	22	29	3230	105	183
3	Acura	TL 4dr	Sedan	Asia	Front	33195	30299	3.2	6.0	270	20	28	3575	108	186
4	Acura	3.5 RL 4dr	Sedan	Asia	Front	43755	39014	3.5	6.0	225	18	24	3880	115	197

Now round-trip the data frame back to a SAS data set.

```
In [23]: cars_full_circle = sas.df2sd(car_df, 'cfc')
```

Figure 3. Transferring Data from Python to SAS Using SASPy

Further, Python methods are available that act as wrappers for SAS code, making the SAS code approachable for someone with a Python programming background while maintaining the power and governance of SAS. A `teach_me_sas` method shows how the Python method creates the SAS code for those who are curious to learn.

Now, let's learn a little SAS. With teach_me_SAS, any of the Python methods that run code will show you the code instead of running it. This way you can copy and paste the SAS output code into a sas.submit() method, change it around, play with syntax, and try your own version of code.

```
In [40]: sas.teach_me_SAS(True)
```

```
In [41]: cars.tail(24)
         proc print data=sashelp.cars(obs=428 firstobs=405 );run;
```

```
In [42]: cars.describe()
         proc means data=sashelp.cars stackodsoutput n nmiss median mean std min p25 p50 p75 max;run;
```

Figure 4: Example of teach_me_sas Method in SASPy

A Jupyter notebook with many SASPy examples is available at GitHub: https://github.com/sassoftware/saspy-examples/blob/master/SAS_contrib/saspy_example_github.ipynb.

SWAT

SWAT is a Python module that interfaces directly into SAS® Cloud Analytic Services (CAS, a part of SAS Viya) for Python 2.7.x or 3.4 and later. SWAT is specifically designed to help Python programmers handle large volumes of data through the scalable architecture of SAS Viya. From SWAT, all CAS actions that you have licensed are available to you via Python methods. You can transfer data to and from CAS directly, so it is possible to mix and match analytic workloads. One of SWAT's major points of emphasis is to mimic the data manipulation syntax of the popular pandas module, which is enabled through CAS table objects that run CAS actions behind the scenes. For portability to other programming clients, you can retrieve the underlying actions that have been run. Figure 5 provides a simple example.

Create a CAStable object and perform pandas-style CAS actions.

In [6]:
```
df = s.CASTable('hmeq')
```

In [7]:
```
type(df)
```

Out[7]:
```
swat.cas.table.CASTable
```

In [8]:
```
df.head()
```

Out[8]:

Selected Rows from Table HMEQ

	BAD	CLAGE	CLNO	DEBTINC	DELINQ	DEROG	JOB	LOAN	MORTDUE	NINQ	REASON	VALUE	YOJ
0	0.0	102.422388	33.0	31.990311	0.0	0.0	ProfExe	20800.0	150507.0	2.0		126763.0	4.0
1	0.0	310.231820	20.0	43.217417	0.0	0.0	Mgr	20800.0	97360.0	1.0	HomeImp	123854.0	0.0
2	0.0	407.585624	24.0	22.162873	0.0	0.0	Sales	20800.0	NaN	0.0	DebtCon	74486.0	6.0
3	0.0	63.248877	23.0	34.599669	0.0	0.0	Office	20800.0	63764.0	0.0	DebtCon	97090.0	1.0
4	0.0	412.014873	23.0	21.945849	0.0	0.0	Sales	20800.0	NaN	0.0	DebtCon	78483.0	2.0

You can retrieve the underlying CAS actions that were performed.

In [9]:
```
s.history(first = -1)
```

```
NOTE: 11: action table.fetch / table={name='hmeq'}, from=1, to=5, sasTypes=false, index=true, _apptag='UI', _messageLevel='erro
r'; /* (SUCCESS) */
```

Figure 5. Example of Printing Table Contents Using SWAT

You can find many GitHub examples to help you get started using SWAT at https://github.com/sassoftware/sas-viya-programming.

ESPPY

ESPPy enables you to programmatically create, modify, and visualize SAS Event Stream Processing components for data transformation, model building, and model scoring via Python objects for Python 2.7.x or 3.4 and later. These objects include components for projects, continuous queries, windows, events, loggers, SAS® Micro Analytic Service modules, routers, and analytical algorithms. Figure 6 shows how a project can be loaded and visualized using the graphviz open-source package.

In [8]:
```
walk = esp.load_project('model_walking.xml')
walk
```

Out[8]:

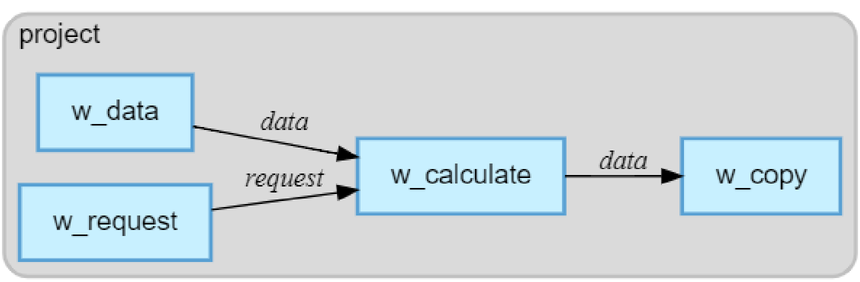

Figure 6. Loading and Visualizing a Project Using ESPPy

Further, models can be trained outside of SAS Event Stream Processing and be used for real-time scoring as in the example at https://github.com/sassoftware/python-esppy/blob/master/examples/Iris_FitStat.ipynb.

In addition to the previously mentioned foundational packages (SASPy, SWAT, ESPPy), SAS provides three higher-level packages (Pipefitter, SASOptPy, and DLPy) that use the foundational packages to help with specific, commonly performed tasks.

PIPEFITTER

Pipefitter is a high-level API that enables you to efficiently build SAS data transformations and machine learning pipelines with a minimal amount of coding for Python 2.7.x or 3.4 and later. It uses either SASPy for SAS 9.4 or SWAT for SAS Viya. Features include imputation, multiple machine learning techniques (including gradient boosting and neural networks), and hyperparameter optimization. Figure 7 shows an excerpt of how simple it is to create and score a decision tree model from the example at https://github.com/sassoftware/python-pipefitter/blob/master/examples/regressionExample.ipynb.

```
In [7]: params = dict(target='label',
                      inputs=['a'+str(i) for i in range(50)])

In [8]: dtree = DecisionTree(max_depth=6, **params)
        dtree

Out[8]: DecisionTree(alpha=0.0, cf_level=0.25, criterion=None, inputs=['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8', 'a9', 'a1
        0', 'a11', 'a12', 'a13', 'a14', 'a15', 'a16', 'a17', 'a18', 'a19', 'a20', 'a21', 'a22', 'a23', 'a24', 'a25', 'a26', 'a27', 'a2
        8', 'a29', 'a30', 'a31', 'a32', 'a33', 'a34', 'a35', 'a36', 'a37', 'a38', 'a39', 'a40', 'a41', 'a42', 'a43', 'a44', 'a45', 'a4
        6', 'a47', 'a48', 'a49'], leaf_size=5, max_branches=2, max_depth=6, n_bins=20, nominals=[], prune=False, target='label', var_imp
        ortance=False)
```

Decision Tree Fit and Score of CAS Table

Using the `DecisionTree` instance, the `fit` method is first run on the data set. This will return a model object.

```
In [9]: model = dtree.fit(casdata)
        model

Out[9]: DecisionTreeModel(alpha=0.0, cf_level=0.25, criterion=None, inputs=['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 'a8', 'a9',
        'a10', 'a11', 'a12', 'a13', 'a14', 'a15', 'a16', 'a17', 'a18', 'a19', 'a20', 'a21', 'a22', 'a23', 'a24', 'a25', 'a26', 'a27', 'a
        28', 'a29', 'a30', 'a31', 'a32', 'a33', 'a34', 'a35', 'a36', 'a37', 'a38', 'a39', 'a40', 'a41', 'a42', 'a43', 'a44', 'a45', 'a4
        6', 'a47', 'a48', 'a49'], leaf_size=5, max_branches=2, max_depth=6, n_bins=20, nominals=[], prune=False, target='label', var_imp
        ortance=False)
```

The `score` method can then be called on the resulting model object.

```
In [10]: score = model.score(casdata)
         score

Out[10]: Target                                          label
         Level                                        INTERVAL
         Var                                      _DT_PredMean_
         NBins                                             100
         NObsUsed                                         1000
         TargetCount                                      1000
         TargetMiss                                          0
         PredCount                                        1000
         PredMiss                                            0
         AverageAbsoluteError                          5.76552
         AverageSquaredError                           52.6174
         AverageSquaredLogarithmicError               0.718349
         RootAverageAbsoluteError                      2.40115
         RootAverageSquaredError                       7.25378
         RootAverageSquaredLogarithmicError           0.847555
         dtype: object
```

Figure 7. Modeling and Scoring a Decision Tree Using Pipefitter

SASOPTPY

SASOptPy provides a Python-friendly way of interacting with SAS/OR® and SAS® Optimization on SAS Viya for Python 3.5 and later; it is specifically designed for linear, mixed integer linear, and nonlinear optimization problems. Like Pipefitter, SASOptPy is built on top of SASPy to run on SAS 9.4 or on top of SWAT to run on SAS Viya. Figure 8 shows an example of a simple linear programming problem that uses SASOptPy.

```python
from swat import CAS
import sasoptpy as so

# Create a CAS Session
s = CAS(hostname='host', port=12345)
# Create an empty optimization model
m = so.Model('demo', session=s)
# Add variables
x = m.add_variable(vartype=so.CONT, name='x')
y = m.add_variable(vartype=so.INT, name='y')
# Set objective function
m.set_objective(2*x+y, sense=so.MAX, name='obj')
# Add constraints
m.add_constraint(x+2*y <= 4.5, name='c1')
m.add_constraint(3*x+y <= 5.5, name='c2')
# Solve the optimization problem
result = m.solve()
# Print and list variable values
print(so.get_solution_table(x, y))
print('Optimal objective value:', m.get_objective_value())
```

Figure 8. Linear Programming Using SASOptPy

You can find many SASOptPy examples in the GitHub repository at https://github.com/sassoftware/sasoptpy/tree/master/examples.

DLPY

DLPy enables programmers of Python 3.4 and later to easily apply SAS Viya deep learning algorithms to image, text, audio, and time series data by using SAS® Visual Data Mining and Machine Learning. DLPy provides high-level APIs for calling deep, convolutional, and recurrent neural networks, with predefined neural network architectures such as VGG-16, ResNet, DenseNet, Darknet, Inception, and YOLO. Figure 9 shows an example of scoring and visualizing a YOLO model that is built using DLPy.

Score the test data, and display object predictions

Now use `predict()` to score the images in the test data `predict_tbl`, and enable use of GPU0 during processing.

In [19]:
```
yolo_model.predict(data=predict_tbl, gpu = Gpu(devices=[0]))
```

```
NOTE: Due to data distribution, miniBatchSize has been limited to 1.
NOTE: Only 1 out of 2 available GPU devices are used.
```

Out[19]: **§ ScoreInfo**

	Descr	Value
0	Number of Observations Read	6
1	Number of Observations Used	0
2	Average IOU in Detection	.

§ OutputCasTables

	casLib	Name	Rows	Columns	casTable
0	CASUSER(ethem-kinginthenorth)	Valid_Res_B3esj1	6	5075	CASTable('Valid_Res_B3esj1', caslib='CASUSER(e...

elapsed 2.56s · user 1.55s · sys 0.893s · mem 2.06e+03MB

This output shows an output CAS table `Valid_Res_B3esj1`, which contains the scored image data that was created.

Now, use `display_object_detections()` with `valid_res_tbl` to create a three-column matrix of scored images, which shows object-detection bounding boxes with label and probability score.

In [20]:
```
display_object_detections(conn=s,
                          coord_type='yolo',
                          max_objects=5,
                          table=yolo_model.valid_res_tbl,
                          num_plot=10,
                          n_col=3)
```

Figure 9. Example of Scoring and Visualizing a YOLO Model by Using DLPy

Another feature in DLPy enables you to import and export deep learning models in ONNX format for portability, as shown in Figure 10.

Load ONNX YOLO Model

```
In [3]: onnx_model = onnx.load('/disk/linux/dlpy/tiny_yolov2/model.onnx')
```

Specify YOLO Anchors and DLPy Detection Layer

```
In [4]: yolo_anchors = (1.08,1.19,  3.42,4.41,  6.63,11.38,  9.42,5.11,  16.62,10.52)

        output_layer = Detection(name='Detection1',
                                 detection_model_type='yolov2',
                                 anchors=yolo_anchors,
                                 predictions_per_grid=5,
                                 class_number=20,
                                 softmax_for_class_prob=True,
                                 object_scale=5.0,
                                 prediction_not_a_object_scale=1.0,
                                 class_scale=1.0,
                                 coord_scale=1.0,
                                 act='LOGISTIC',
                                 grid_number=13,
                                 coord_type='YOLO',
                                 detection_threshold=0.3,
                                 iou_threshold=0.3)
```

Convert ONNX Model to DLPy Model, and Generate H5 Weights File

```
In [5]: model1 = Model.from_onnx_model(conn=s,
                                       onnx_model=onnx_model,
                                       output_model_table='tiny_yolov2',
                                       scale=1/255.,
                                       output_layer=output_layer)

        NOTE: Successfully written weights file as /root/working/tiny_yolov2_weights.onnxmodel.h5
        NOTE: Model table is attached successfully!
        NOTE: Model is named to "tiny_yolov2" according to the model name in the table.
        NOTE: Successfully imported ONNX model.
```

Figure 10. Loading a YOLO Model in ONNX format by Using DLPy

You can find examples for these features in the GitHub repository at https://github.com/sassoftware/python-dlpy/tree/master/examples.

SAS KERNEL FOR JUPYTER NOTEBOOK

To enable you to access SAS analytics from open-source interfaces, SAS also supports a popular notebook environment for programming: the SAS kernel for Jupyter Notebook. It requires Python 3.x, Jupyter 4 or later, and SAS 9.4 or SAS Viya. Behind Jupyter Notebook is a Python session that submits code to SAS and receives responses through a socket interface. A notebook-style approach to SAS programming enables you to interactively submit code and visualize responses as seen in Figure 11.

Run SAS Code in Jupyter Notebook!

Print the first few rows

In [1]:
```
proc print data = sashelp.cars (obs=10);
run;
```

Out[1]:

The SAS System

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197

Run PROC MEANS

In [2]:
```
proc means data = sashelp.cars;
run;
```

Out[2]:

The SAS System

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
MSRP		428	32774.86	19431.72	10280.00	192465.00
Invoice		428	30014.70	17642.12	9875.00	173560.00
EngineSize	Engine Size (L)	428	3.1967290	1.1085947	1.3000000	8.3000000
Cylinders		426	5.8075117	1.5584426	3.0000000	12.0000000
Horsepower		428	215.8855140	71.8360316	73.0000000	500.0000000
MPG_City	MPG (City)	428	20.0607477	5.2382176	10.0000000	60.0000000
MPG_Highway	MPG (Highway)	428	26.8434579	5.7412007	12.0000000	66.0000000
Weight	Weight (LBS)	428	3577.95	758.9832146	1850.00	7190.00
Wheelbase	Wheelbase (IN)	428	108.1542056	8.3118130	89.0000000	144.0000000
Length	Length (IN)	428	186.3621495	14.3579913	143.0000000	238.0000000

Figure 11. Executing SAS Code in Jupyter Notebook Using the SAS Kernel

To improve usability, SAS extensions and Jupyter magic commands are also available in the GitHub repository at https://github.com/sassoftware/sas_kernel.

OPEN SOURCE TO SAS LANGUAGE AND INTERFACES

Next, let's see how SAS enables bringing models from open-source software into SAS 9 or SAS Viya.

In this case, models are trained and scored in the open-source software. A scored data set or the PMML (Predictive Model Markup Language) score code produced by the open-source software is passed to the SAS platform to compute model assessment and perform model comparison. The scored data set contains model predictions for an interval target or the posterior probabilities for a nominal target. When PMML score code instead of a scored data set is passed, SAS converts this score code into DATA step scoring logic and supports model deployment. This deployment allows supported models to be published to databases such as Hadoop, Teradata and so on or to be registered to a repository for future model management by products such as SAS® Model Manager.

The interfaces in this category can be classified as based either on programming or on a GUI (graphical user interface):

- Programming interfaces on SAS 9:
  - SAS/IML®
  - Base SAS® Java Object
- GUI-based:
  - SAS® Enterprise Miner™ (Open Source Integration node for R, SAS Code node for Python) on SAS 9
  - Model Studio in SAS Visual Data Mining and Machine Learning (Open Source Code node) on SAS Viya

PROGRAMMING: SAS/IML

SAS/IML software provides a flexible matrix programming language that enables statistical programmers to perform data analysis, simulation, and matrix computations. It also includes the ability to call functions in the R language from within the IML procedure in SAS/IML 9.22 and later. To use this ability, R software must be installed on the SAS Workspace Server and the SAS system must have been configured with the RLANG option.

The SAS/IML functions ImportDataSetFromR, ExportDataSetToR, ImportMatrixFromR, ExportMatrixToR make it easy to transfer data between SAS and R data structures. You specify the R code that needs to be executed between the SUBMIT / R and ENDSUBMIT statements within PROC IML. To get started on this methodology, see the section "Calling Functions in the R Language" in *SAS/IML 15.1: User's Guide*. The following statements provide an example that uses PROC IML to perform linear regression in R software:

```
/* Use PROC IML to build a regression model in R software */
proc iml;
  call ExportDataSetToR("sashelp.class", "class");
  submit / R;
    Model <- lm(Weight ~ Height, data=class, na.action="na.exclude")
    ParamEst <- coef(Model)
  endsubmit;
  call ImportDataSetFromR("work.ParamEst", "ParamEst");
quit;

/* Print intercept and height parameter estimates */
proc print data=work.ParamEst;
run;
```

PROGRAMMING: BASE SAS JAVA OBJECT

The Java Object functionality in Base SAS (9.2 and later) provides a mechanism that is like the Java Native Interface (JNI) for instantiating Java classes and accessing fields and methods on the resultant objects. Using the Java Object, you can create hybrid applications that bring together the capabilities of both SAS and Java.

With the Base SAS Java Object, the communication between SAS and the open-source software is through a pair of Java classes available in the GitHub repository at https://github.com/sassoftware/enlighten-integration/tree/master/SAS_Base_OpenSrcIntegration in the src/dev folder. After you compile the provided Java classes and set the Java CLASSPATH accordingly in your Base SAS installation, you can use the SAS DATA step code to instantiate the Java class to execute the Python or R program.

The following SAS DATA step instantiates the Java Object and executes a method:

```
*** PYTHON and WORK_DIR LOCATIONS (-- USER UPDATE NEEDED --);
%let PYTHON_EXEC_COMMAND=C:\Anaconda\python.exe;
%let WORK_DIR=C:\SAS_Base_OpenSrcIntegration;

/* Executing Python code */
data _null_;
  *** Python program takes working directory as first argument;
  python_call = "&WORK_DIR.\digitsdata_svm.py &WORK_DIR";
  declare javaobj j("dev.SASJavaExec", "&PYTHON_EXEC_COMMAND", python_call);
  j.callStaticVoidMethod("main");
  j.delete();
run;
```

This technique can be used to invoke any external program, whether Python, R, MATLAB, or others. However, that type of flexibility can also bring security concerns depending on which users and how many of them can access the SAS server machine.

The step-by-step details about implementing this methodology can be found in Hall, Myneni, and Zhang (2015), which includes Python and R code samples.

GUI: SAS ENTERPRISE MINER

The Open Source Integration node was introduced in SAS Enterprise Miner 13.1 to enable you to write code to train models in the R language and when possible support scoring of these supervised or unsupervised models alongside SAS models. The node's **Training Mode** and the **Output Mode** properties control how modeling results and generated columns from R software are returned to and consumed by SAS Enterprise Miner. With regard to configuration and setup, R software must be installed on the same machine where SAS Enterprise Miner is installed, and all necessary packages must be pre-installed before they are used in the node.

In the Open Source Integration node, the **Training Mode** property specifies whether a supervised or unsupervised model is being built, and the **Output Mode** property specifies whether the R code returns output according to the **PMML** option (which returns PMML score code), **Merge** option (which returns model predictions), or the **None** option (which returns no output). The PMML output mode enables certain standard R packages (such as lm, multinom(nnet), glm(stats), rpart, kmeans(stats), and nnet) to generate PMML score code that can be turned into SAS score code. When this is possible, these R models can also be deployed to databases (in order to score new data) or registered to SAS Model Manager (for further monitoring and management).

The Open Source Integration node relies on the SAS/IML integration with R under the covers; that is, it requires that the SAS system was configured with the RLANG option although SAS/IML does not have to be licensed separately. Figure 12 shows an example process flow diagram from *SAS Enterprise Miner 15.1: Reference Help* that illustrates how to use R software to model and score a logistic regression for a binary target.

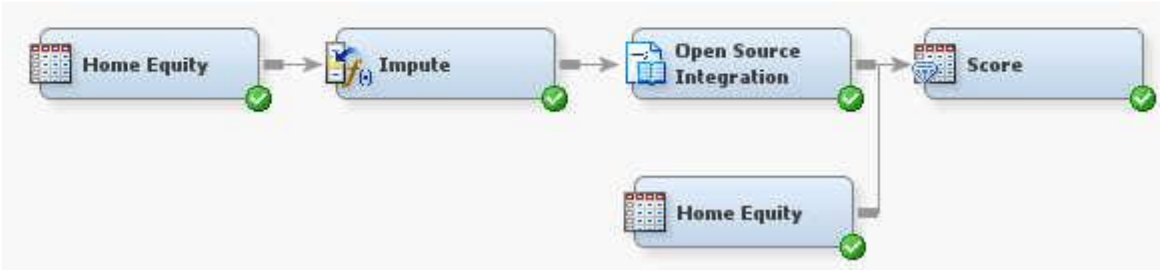

Figure 12. SAS Enterprise Miner Process Flow Diagram That Uses the Open Source Integration Node

The following logistic regression R code (which you specify in the Open Source Integration node) is relatively straightforward when you use data and variable handles that are provided by the node.

```
&EMR_MODEL <- glm(&EMR_CLASS_TARGET ~ &EMR_CLASS_INPUT + &EMR_NUM_INPUT,
family= binomial(), data= &EMR_IMPORT_DATA)
```

For detailed steps on re-creating this example, see *SAS Enterprise Miner 15.1: Reference Help*.

Although Python is not supported in the Open Source Integration node, it can be executed using the SAS Code node and the Java Object functionality in Base SAS. The following SAS Communities tip shows how to execute a Python script in SAS Enterprise Miner: https://communities.sas.com/t5/SAS-Communities-Library/Tip-How-to-execute-a-Python-script-in-SAS-Enterprise-Miner/ta-p/223761

GUI: MODEL STUDIO IN SAS VISUAL DATA MINING AND MACHINE LEARNING

Just as SAS Enterprise Miner provides the Open Source Integration node, the Model Studio application in SAS Visual Data Mining and Machine Learning 8.3 and later provides the Open Source Code node, which can execute Python or R code.

The Open Source Code node (located in the Miscellaneous group) can train a Python or R model that can subsequently be assessed and compared with other SAS, Python, or R models in the Model Studio pipeline. A Model Studio pipeline enables you to perform a series of tasks (such as data preprocessing, feature engineering, predictive modeling, data postprocessing, and model ensembles) followed by comparison of these models in a directed process flow. These tasks, called "nodes" in Model Studio, provide a large choice of statistical, data mining, machine learning, model interpretation, and deployment techniques for analyzing your data. Because SAS Visual Data Mining and Machine Learning runs in SAS Viya, it can handle large amounts of data using in-memory, distributed computing techniques.

To use the Open Source Code node, Python or R must be installed on the same machine as the compute server microservice. On Linux, the executable `python` or `Rscript` must be available in the system path. If you have multiple versions of Python or R on your compute server, you can set a preferred version by modifying the PATH environment variable. You also need to install any necessary Python or R packages with administrator or sudo privileges so that they are accessible to all users.

Figure 13 shows an example Model Studio pipeline that trains and compares various forest models from SAS, Python, and R software. For more information about this pipeline, see the SAS Communities Library: https://communities.sas.com/t5/SAS-Communities-Library/How-to-execute-Python-or-R-models-using-the-Open-Source-Code/ta-p/499463. The nodes in

Figure 13 depict the following, from left to right (the last three of these nodes are Open Source Code nodes):

- Forest node in Model Studio
- randomForest package in R
- scikit-learn RandomForestClassifier in Python
- scikit-learn RandomForestClassifier in Python, where categorical inputs are one-hot encoded

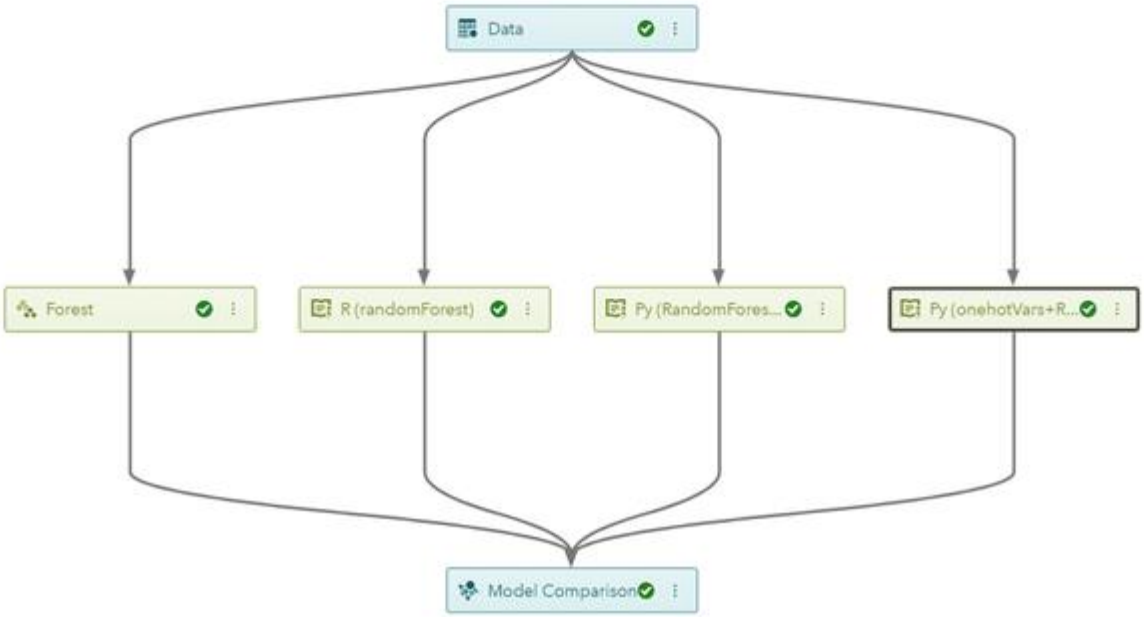

Figure 13. Model Studio Pipeline That Compares Multiple Models by Using the Forest Node and Open Source Code Nodes

Although not shown in this example, you can add other preprocessing nodes such as Feature Extraction, Filtering, Imputation, Transformations, Variable Selection, and so on as needed after the Data node and before any Open Source Code nodes in this pipeline. Note that any output file from Python or R code that is prefixed with rpt_ and saved as a comma-separated value file (with a .csv file extension), as a plain text file (with a .txt file extension), or as an image file (with .png, .jpeg, or .gif file extensions) can be viewed in the Results view of the Open Source Code node after execution.

For more information about the inner workings of the Open Source Code node, see the node documentation at https://go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.3&docsetId=vdmmlref&docsetTarget=n0gn2o41lgv4exn17lngd558jcso.htm&locale=en and examples in the GitHub repository at https://github.com/sassoftware/sas-viya-dmml-pipelines/tree/master/open_source_code_node. In addition, you can view a brief video on this topic at http://video.na.sas.com/assets/47183.

CONCLUSION

Providing integration with open-source tools such as Python and R is a major focus area for SAS. Whether it is bringing SAS analytics to open source or open-source capabilities into SAS software, SAS recognizes that enabling and enhancing integration points between various software systems improves approachability, collaboration, and time-to-value.

REFERENCES

SAS Institute Inc. 2018. *SAS/IML 15.1: User's Guide*. Cary, NC: SAS Institute Inc. Available: https://go.documentation.sas.com/?docsetId=imlug&docsetTarget=imlug_r_toc.htm&docsetVersion=15.1&locale=en

Hall, P., Myneni, R., and Zhang, R. 2015. "Open Source Integration Using the Base SAS Java Object". SAS Institute Inc. Available: https://github.com/sassoftware/enlighten-integration/blob/master/SAS_Base_OpenSrcIntegration/SAS_Base_OpenSrcIntegration.pdf

SAS Institute Inc. 2018. *SAS Enterprise Miner 15.1: Reference Help*. Cary, NC: SAS Institute Inc. Available: https://documentation.sas.com/?docsetId=emref&docsetTarget=bookinfo.htm&docsetVersion=15.1

SAS Institute Inc. 2018. *SAS Visual Data Mining and Machine Learning 8.3: User's Guide*. Cary, NC: SAS Institute Inc. Available: https://go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.3&docsetId=vdmmlug&docsetTarget=titlepage.htm&locale=en

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Jesse Luebbert
jesse.luebbert@sas.com

Radhikha Myneni
radhikha.myneni@sas.com