Paper SAS4440-2020

Scalable Cloud-Based Time Series Analysis and Forecasting Using Open-Source Software

Javier Delgado, Thiago Quirino, and Michael Leonard, SAS Institute Inc.

ABSTRACT

Many organizations need to process large numbers of time series for analysis, decomposition, forecasting, monitoring, and data mining. The TSMODEL procedure, available in SAS® Visual Forecasting and SAS Econometrics® software, provides a resilient, distributed, and optimized generic time series analysis environment for cloud computing. PROC TSMODEL offers capabilities such as automatic forecast model generation, automatic variable and event selection, automatic model selection, and parameter optimization. It also provides advanced support for time series analysis (in the time domain or in the frequency domain), time series decomposition, time series modeling, signal analysis and anomaly detection (for IoT), and temporal data mining. In addition, PROC TSMODEL supports open-source integration with external languages Python and R. This paper describes the scripting language that supports cloud-based open-source integration between SAS® software and external languages; examples that demonstrate this use case are provided.

INTRODUCTION

More information than ever before is being collected with associated timestamps. Computers, mobile phones, smart devices, detectors, and other devices record timestamped data. These timestamped data can be modeled, forecasted, or mined (or any combination of these) for better decision-making. In most cases, the decisions are critical and have immense financial and ethical implications. For example:

- Retailers rely on both seasonal and nonseasonal forecasts of product demand in order to make profitable decisions about staff scheduling and stocking levels for millions of products across thousands of stores.
- Manufacturers rely on accurate forecasts of time to component failure in order to make decisions about the maintenance schedule of critical machinery components.
- Railroad companies rely on accurate time series forecasts of shipping demand per region of the country in order to preemptively stock their railroad cars across different regions. Accurate forecasts enable them to better meet the predicted demand, minimize shipping delays, and improve customer satisfaction.
- Energy companies rely on the ability to both monitor and analyze, in real time, sensor data that stream from wind turbines. Time series of sensor data are analyzed in order to quickly detect and respond to critical anomalous behavior and to maintain their turbines at peak performance over time.
- Hospitals can aggregate patient sensor data, lab results, and physician notes in order to monitor patient progress and better predict patient outcome. Similarly, a physician can monitor a patient's pacemaker remotely in order to quickly determine when the patient's heart is behaving anomalously.
- Governments rely on time series decomposition techniques in order to decompose series of economic variables into their long-term trends and short-term seasonal effects so that they can gain a better insight into the real status of the economy.

In recent years, there has been an enormous increase in the amount of timestamped data being collected. It is now commonplace for companies (such as banks, manufacturers, retailers, websites, hospitals, universities, and governments, in addition to taxi, insurance, stock trading, phone, energy, and many more companies) to maintain large databases of timestamped data whose sizes range from hundreds of gigabytes to hundreds of terabytes. These databases are gold mines for insights into consumer behavior. These insights can help organizations optimize their internal processes to better meet consumer demands.

The amount of timestamped data being collected is expected to further escalate because of the ongoing proliferation of the Internet of Things (IoT). IoT enables all types of objects (cars, toasters, pacemakers, water and gas meters, and so on) to be discovered, monitored, and controlled remotely via the existing internet infrastructure. In short, "big data" has become pervasive in today's society: it is everywhere and in anything, it is here to stay, and it has a lot to say. Processing this ever-increasing amount of timestamped data in an intelligent way poses both architectural and analytical challenges. For example, because of the sheer amount of data and the ever-increasing demand to gain decision-making insights from data in close to real time, time series analysis of big data is inherently a distributed computing problem and is thus an architectural challenge. In addition, big data solutions must be generic enough to accurately handle the time series analysis requirements of different applications and thus are an analytical challenge.

SAS Visual Forecasting provides procedures for some of the most common analyses that are performed on timestamped data: forecasting, decomposition and price analysis, time series monitoring and anomaly detection, and temporal data mining. This paper provides an overview of the SAS Visual Forecasting procedures—in particular of the TSMODEL procedure, which was specifically designed to support advanced, efficient, and cloud-based time series analysis of big data. Particular emphasis is given to integrating Python and R code with PROC TSMODEL in order to enable efficient, massively parallel execution of Python and R programs.

HOW THE TSMODEL PROCEDURE WORKS

The goal of cloud-based time series analysis and forecasting is to perform an analytical task in a single pass through the data by using a distributed file system or distributed computing environment (or both). Moving data can strain computing resources, whether internal to a node, external (between computing nodes), or both. A single pass through the data allows for enormous performance gains. By providing a system that both moves data and computes efficiently, the TSMODEL procedure makes time series analysis and forecasting possible on an enormous scale. PROC TSMODEL procedure provides a scalable, cloud-based time series analysis environment, which includes a distributed file system, a scripting environment, and parallel data reading, script execution, and data writing. It is designed to run in the SAS Cloud Analytic Services (CAS) run-time environment that is deployed with SAS Visual Forecasting. The following sections describe these elements in more detail.

DISTRIBUTED FILE SYSTEM

PROC TSMODEL is designed to enable your analysis to use a distributed file system (DFS). A DFS allows for redundant and resilient storage of data; it breaks up large files into chunks and stores each chunk on several storage media. In addition, it makes several redundant copies of each chunk in order to forgo the need for making periodic backup copies. If a particular file system fails, the distributed file system can resiliently heal itself without needing to restore backup copies (which could cause delays). However, the data are not stored contiguously in such a file system, so sorting on a particular file system is not possible. This is particularly problematic for time series analysis, where the ordering of the data is crucial. In addition, the data that are needed for time series analysis might be stored in several files. These distributed files must be read, sorted, and merged with respect to

time in a scalable and efficient way. SAS Visual Forecasting procedures automatically perform all these operations on the input time series data in preparation for the analysis.

Figure 1 illustrates a cluster that consists of four worker nodes and a distributed file system that contains two tables, A and B. Each table is organized by classification (BY) variables that delineate the time series rows, which are grouped into seven BY groups. Each BY group represents one time series. One or more computing (worker) nodes are connected to the distributed file system; neither the tables nor the BY groups are stored on a single machine.

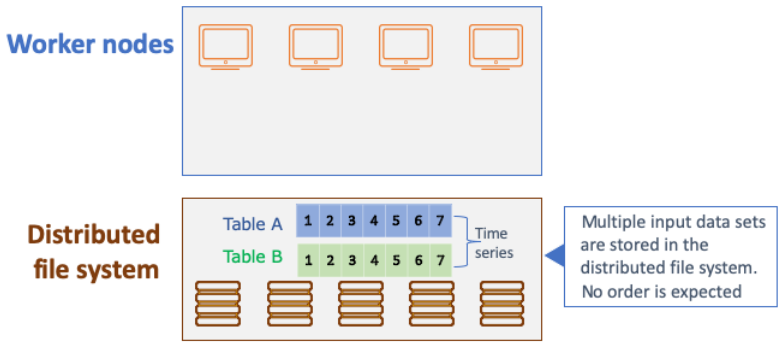


Figure 1. Distributed File System

SCRIPTING LANGUAGE, DISTRIBUTION, AND COMPILATION

The vast amount of data that cloud computing can support calls for a time series analysis environment that allows data to be processed efficiently. SAS Visual Forecasting provides a scripting language that facilitates the use of various capabilities, such as the following:

- automatic forecast model generation, automatic variable and event selection, automatic model selection, and parameter optimization
- advanced support for time series analysis (in the time domain or in the frequency domain), time series decomposition, time series modeling, signal analysis and anomaly detection (for IoT), and temporal data mining
- preparation of the input data prior to analysis and postprocessing of the final results in the same script
- reading of multiple input data files and creation of multiple output data files

These features make the scripting language flexible and useful for numerous applications. Figure 2 illustrates the use of this scripting language. The script is created outside the computing server and can be submitted to the server by SAS, Python, Lua, or R clients.

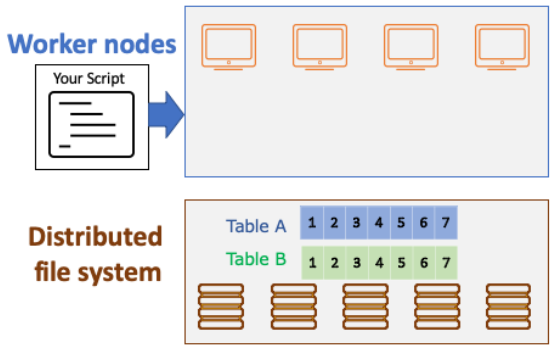


Figure 2. Scripting Language: User Script Contains SAS Code and Optionally Python and R Code

The distributed network can consist of one or more computing (worker) nodes. After being submitted to the computing server, the user-specified script is distributed to each worker node to permit parallel execution of the specified analysis, as shown in Figure 3.

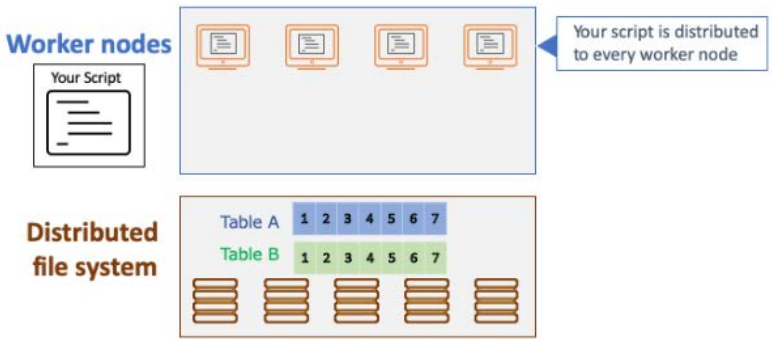
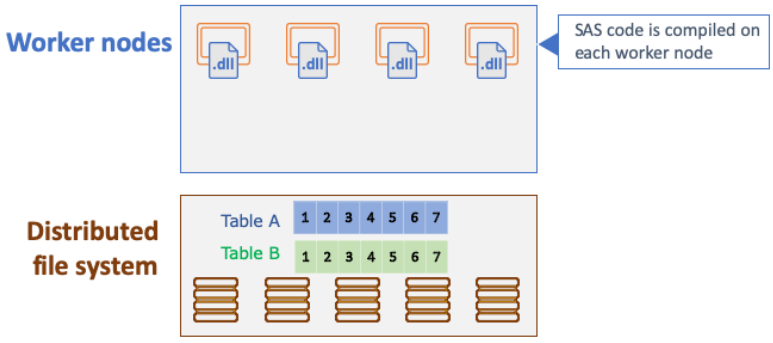


Figure 3. Script Distribution

The user-specified script is then compiled on each of the computing nodes. The compiler optimizes the resulting executable for the specific operating system of the computing node (Linux, Windows, and so on). This optimized executable permits very fast execution of the specified analysis. Any external language source code you included in the script is stored in memory. One or more external language interpreters are launched for each thread on each worker node in order to process the external language code at run time.

Figure 4 illustrates the script compilation and execution process when only SAS code is run and when external-language code is integrated. After the script is distributed to the computing (worker) nodes, it is optimally compiled.

(a) User script contains only SAS code:



(b) User script contains SAS and Python code:

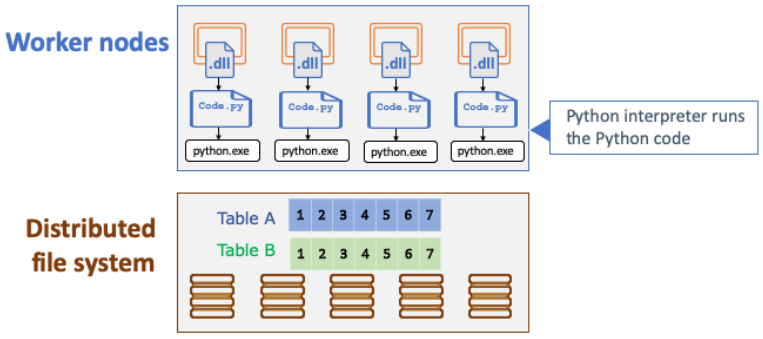


Figure 4. Script Compilation (a) without and (b) with External-Language Code

PARALLEL READ

All the computing nodes read one or more input data files simultaneously. Each input data file contains unsorted, timestamped transactional data that might be recorded at no fixed interval. However, time series analysis algorithms typically require that the input time series data be stored contiguously in memory, in temporal order, and with a fixed-time interval. Therefore, the transactional data must be transformed into a suitable form prior to analysis. PROC TSMODEL relies on the properties of the input data in order to determine how to transform the data for optimal performance. For example, when the input data consist of multiple time series (BY groups), then the transformation occurs via a two-step process that is illustrated in Figure 5 and described in detail in the following sections.

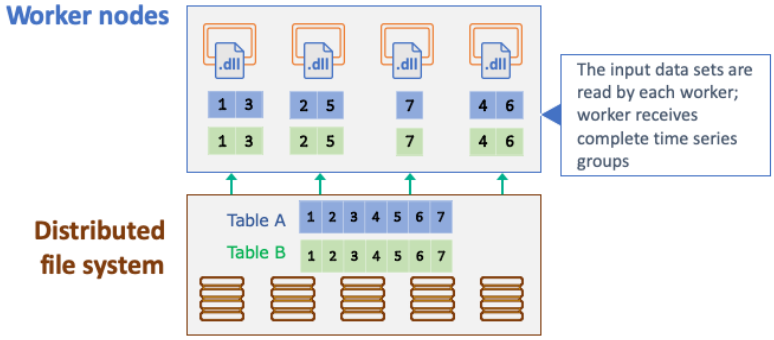


Figure 5. Parallel Read

PARALLEL AND THREADED EXECUTION

Each computing node executes (in parallel) the compiled, optimized script for each time series that has been assigned to it. Each time series is executed on one thread of the computing node. Each of the computing node's threads is kept busy until all the time series that have been assigned to it have been processed. If any problems occur during the execution of a particular time series (BY group), they are logged into an in-memory table so that you can investigate them further. Figure 6 illustrates the parallel execution.

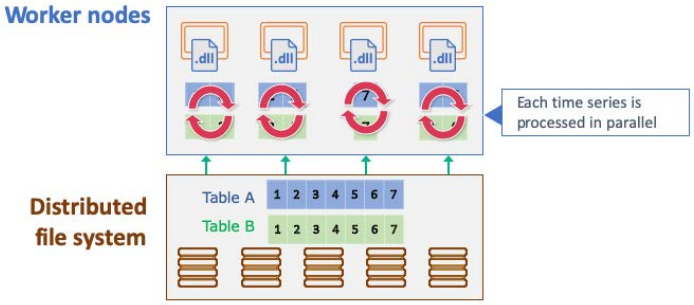


Figure 6. Parallel Execution

PARALLEL AND THREADED EXTERNAL LANGUAGE EXECUTION

The External Languages (EXTLANG) package enables execution of Python and R scripts within the PROC TSMODEL infrastructure. The external-language interpreter is run on the same CAS worker thread where the BY groups data reside, so there is no need for additional internode data transfer. Data are transferred within the worker node and between the SAS process and the external-language interpreter process. Although transfers are backed by a path on disk, the operating system typically uses an in-memory copy of the data, bypassing the need to read the data from disk. On our cluster, we observed a transfer overhead below 2 milliseconds when working with BY-group data sizes of less than 10,000 elements.

PARALLEL WRITE TO THE DISTRIBUTED FILE SYSTEM

After the specified analysis is executed for a particular time series, the computing nodes write one or more output data sets asynchronously and independently. Multiple output data files can be created simultaneously.

Figure 7 illustrates the parallel write. Each time series analysis result is written back to the distributed file system.

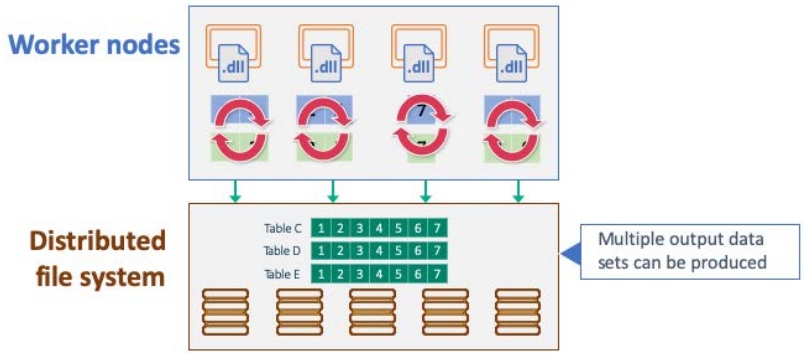


Figure 7. Parallel Write

For more information about scalable cloud-based time series analysis and forecasting, see Quirino, Leonard, and Blair (2018).

IMPLEMENTATION

SAS Visual Forecasting enables you to use a variety of methods (procedures, scripts, packages, and actions) to implement solutions to your time series forecasting problems.

THE TSMODEL PROCEDURE

The TSMODEL procedure is a SAS® Viya® procedure that executes user-defined programs (scripts) on time series data. PROC TSMODEL analyzes timestamped transactional data with respect to time and accumulates the data into a time series format.

PROC TSMODEL forms time series from timestamped transactional input data and writes the accumulated time series variables to an output table. Time series are delineated by distinct values of the variables that are specified in the BY statement.

Timestamped transactional data are not usually recorded at a fixed interval. Because time series analysis techniques often require fixed-time intervals, the transactional data must be transformed into a fixed-interval time series, such as daily, weekly, or monthly.

PROC TSMODEL forms time series vectors from timestamped data and then provides these vectors as array variables for subsequent processing by program statements, which constitute a script. The script is processed independently for each BY group. The syntax of PROC TSMODEL is the same as that of the TIMEDATA procedure, which is similar to the SAS DATA step for time series data. The SAS DATA step processes data row by row, whereas PROC TSMODEL processes time series vectors (columns) for the BY groups.

For more information about PROC TSMODEL, see *SAS Visual Forecasting: Forecasting Procedures*.

SCRIPTS

Scripts consist of statements that perform the desired analysis on each time series. For more information about the object-oriented scripting language that PROC TSMODEL supports, see the FCMP procedure in *Base SAS® Procedures Guide*.

PACKAGES

Packages contain computational services that can be used in your script. A package is a set of related specialized objects and functions (called "methods"), each of which addresses a unique facet of the time series analysis problem. You can use specialized objects and functions to write custom SAS code in order to gain access both to cutting-edge data analysis tools and to utilities that are designed to significantly speed up code development and optimize code quality. Table 1 shows the packages available for PROC TSMODEL.

Package Name	Description
SFS	*Simple Forecast Service:* Tools for automatic forecasting of time series with a simple-to-use interface; these tools use only exponential smoothing (ESM) and ARIMA models
ATSM	*Automatic Time Series Modeling And Forecasting:* Tools for automatic modeling and forecasting of time series by using various model families such as exponential smoothing (ESM), ARIMA, intermittent demand (IDM), and unobserved component (UCM) models
TSA	*Time Series Analysis:* Tools for efficient statistical analysis of time series (transformations, decompositions, statistical tests for intermittency, seasonality, stationarity, forecast bias, and so on)
TSD	*Time Series Distance Measures:* Tools for efficient measure of the distance between two time series or among sequences in temporal data (dynamic time warping, longest common subsequence, and so on)
TDR	*Time Series Dimension Reduction:* Tools for efficient time series dimension reduction (symbolic aggregate approximation, discrete Fourier transformation, discrete wavelet transformation, random projection, singular value decomposition)
TFA	*Time-Frequency Analysis:* Tools for efficient analysis of time series in both time domain and frequency domain
TSM	*Time Series Modeling:* Tools for efficient time series modeling and forecasting
SSA	*Singular Spectrum Analysis:* Tools for decomposing a time series into additive components and categorizing those components on the basis of the magnitudes of their contributions
MSSA	*Multivariate Singular Spectrum Analysis:* Tools for decomposing one or more time series into additive components and categorizing those components on the basis of the magnitudes of their contributions
MTF	*Time Series Motif Discovery:* Tools for the discovery of frequent patterns or repeated subsequences in time series
SST	*Subspace Tracking:* Tools for the analysis and decomposition of time series for tracking and monitoring purposes
TIMFIL	*Time Series Filters:* Tools for performing various types of filtering and aggregation on time series data
UTL	*Utility:* Tools for performing basic statistical computations on pairs of actual and predicted time series
EXTLANG	*External Languages:* Tools for enabling seamless integration of external language programs into SAS environments

Table 1. Packages Available for the TSMODEL Procedure

Some of these packages were developed as cloud-based analogues of traditional SAS products and procedures. For example, the ATSM, SFS, and TSM packages carry the features available in SAS® Forecast Server. Similarly, the TSA, TFA, and SSA packages carry various features that are available in SAS/ETS®, albeit with a different scope. For more information about these packages, see *SAS Visual Forecasting: Time Series Packages*.

ACTIONS

Actions are executed on the CAS workers, using clients available for a variety of languages: SAS, Python, R, and Lua. For more information about actions, see *SAS Visual Forecasting: Programming Guide*.

OPEN-SOURCE INTEGRATION

In addition to being able to execute actions via SAS, Python, Lua, and R clients, the TSMODEL procedure can now execute R and Python scripts via actions that run on the distributed computing servers (that is, the worker nodes in Figure 4). The computational objects provided by the EXTLANG package to facilitate running Python and R programs on the computational servers are summarized in Table 2. Figure 8 illustrates the object data flow diagram for the EXTLANG package. For more information about the EXTLANG package and other packages, see *SAS Visual Forecasting: Time Series Packages*.

Interpreter Object	Description
PYTHON2	Provides support for running code that is written in version 2 of the Python programming language
PYTHON3	Provides support for running code that is written in version 3 of the Python programming language
R	Provides support for running code that is written in the R programming language
Output Object	Description
OUTEXTCODE	Stores user-supplied external-language source code that is supplied via a PYTHON2, PYTHON3, or R object in a CAS table to "replay" it later
OUTEXTLOG	Stores execution and resource usage logs in a table that resides in CAS (a CAS table)
OUTEXTVARSTATUS	Collects the status flags of all shared variables and stores them in a CAS table
Input Object	Description
INEXTCODE	Reads code from a CAS table and provides it to the external language interpreter for reuse on a per-BY-group basis

Table 2. Computational Objects of the EXTLANG Package

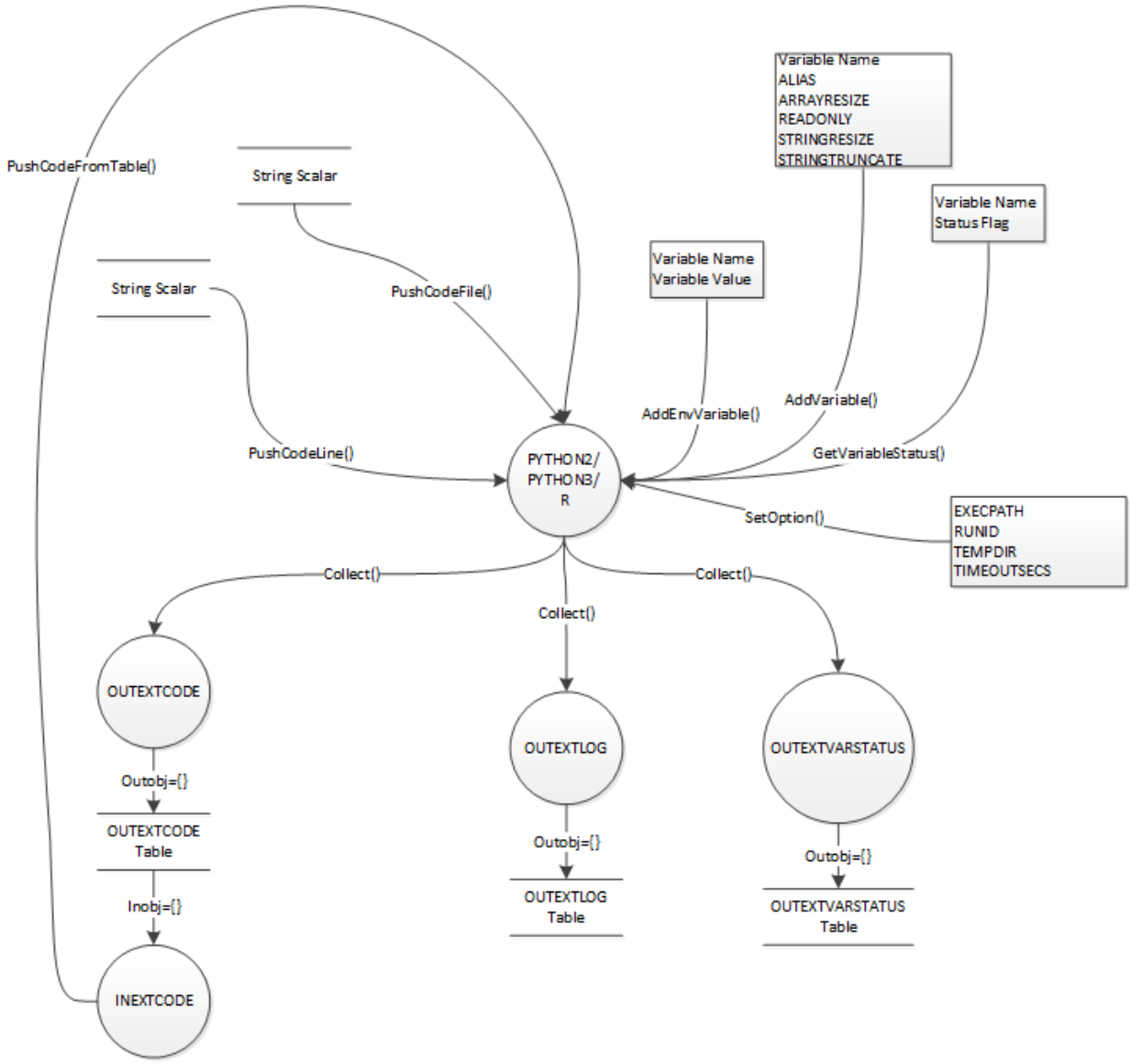


Figure 8. Object Data Flow Diagram for the EXTLANG Package

EXAMPLES

This section provides three examples that demonstrate the capabilities of the TSMODEL procedure, with a specific emphasis on integrating external language programs. The first example illustrates how you can input Python code directly into your SAS script to calculate a simple moving average. The second example shows you can fit an ARIMA model that is implemented in R to your SAS data set. The third example shows the procedure's ability to perform fast time series analysis on big data. All examples use a SAS script as the client to run actions on the CAS server, but you can use any supported CAS client, including Python, R, or Lua.

EXAMPLE 1: MOVING AVERAGE USING PYTHON

This example demonstrates how to use the EXTLANG package to calculate a simple moving average of a SAS data set (Sashelp.PriceData) and transfer the value back to your SAS program. This data set consists of simulated monthly sales data that are hierarchically organized by region, line, and product. The Sashelp.PriceData data set contains 1,020

observations, which are divided into 17 BY groups. By virtue of running within PROC TSMODEL, each BY group is processed in parallel in a separate thread on each worker in the CAS cluster. Although this data set is relatively small, the sample code in this section can also readily handle time series data sets that contain millions of BY groups.

First, a connection to CAS (that is, a session) is established, and a CAS library called *mycas* is created. The *mycas* library enables you to transfer data sets to the CAS nodes, where the distributed time series analysis is performed.

```
CAS mycas;
LIBNAME mycas CAS SESSREF = mycas;
```

A DATA step transfers the *Sashelp.PriceData* data set into the CAS *mycas* library:

```
DATA mycas.pricedata;
  SET sashelp.pricedata;
RUN;
```

The PROC TSMODEL statement specifies the input data set (*mycas.pricedata*), an output table in which to store the output data array (*mycas.outarray*), an output table in which to store the scalar variables (*mycas.outscalar*), and an output table in which to store an output object (*mycas.pylog*).

```
PROC TSMODEL DATA=mycas.pricedata OUTARRAY=mycas.outarray
             OUTSCALAR=mycas.outscalar
             OUTOBJ=(pylog=mycas.pylog);
```

The ID statement specifies the variable `date` as the time index variable, and the INTERVAL= option indicates that the data are monthly.

```
ID date INTERVAL = MONTH;
```

The BY statement specifies that each unique combination of the data set variables `regionname`, `productline`, and `productname` corresponds to a unique time series BY group. BY groups are processed independently.

```
BY regionname productline productname;
```

The VAR statement specifies the input data set variable SALE. The ACCUMULATE=AVG option specifies an average value accumulation for the SALE variable.

```
VAR SALE / ACCUMULATE = AVG;
```

The OUTSCALAR statement specifies the scalar variables that the SAS script is to generate and store. These include a variable in which to store the Python program's execution time (`runtime`), a variable in which to store the exit code (`exitCode`), and variables in which to store the return code from each PYTHON2 object's method call (`rc1–rc6`).

```
OUTSCALAR runtime exitCode rc1 rc2 rc3 rc4 rc5 rc6;
```

The OUTARRAY statement specifies the array variables that the program is to generate and store. The only output array is the moving average (MAVG).

```
OUTARRAY MAVG;
```

The REQUIRE statement specifies the EXTLANG package, which includes all the objects that PROC TSMODEL needs in order to interact with external languages.

```
REQUIRE EXTLANG;
```

The program statements between the SUBMIT and ENDSUBMIT statements use the ATSM package objects to perform the actual analysis on the CAS server:

```
SUBMIT;

    /*
     * Initialize the PYTHON2 object, which is the interface to the
     * Python interpreter.
     */
    declare object py(PYTHON2);
    rc = py.Initialize();

    /*
     * Create the Python program, which simply does the following:
     *  1. Import the NumPy package with alias np
     *  2. Create an array to be used for the moving average computation,
     *     with a window size of 3
     *  3. Compute the moving average and store into variable MAVG
     */
    rc1 = py.PushCodeLine('import numpy as np');
    rc2 = py.PushCodeLine('w = np.ones((3,))/3 ; ');
    rc3 = py.PushCodeLine('MAVG = np.convolve(SALE, w, mode="same")');

    /*
     * Specify variables to share between SAS and Python.
     * The variable SALE is used only as input in the Python program;
     * the default value of READONLY is used to avoid propagating
     * its data back to SAS. MAVG is transferred back to SAS, where
     * it is stored in a CAS table for further analysis.
     */
    rc4 = py.AddVariable(SALE) ;
    rc5 = py.AddVariable(MAVG, 'READONLY', 'FALSE');


    /*
     * Run the program and obtain the run time and exit code.
     */
    rc6 = py.Run();
    runtime = py.GetRuntime();
    exitCode = py.GetExitCode() ;

    /*
     * Store the execution and resource usage statistics logs.
     */
    declare object pylog(OUTEXTLOG);
    rc = pylog.Collect(py,'ALL');

ENDSUBMIT;
RUN;
```

The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Output 1. This summary includes the number of BY groups that are processed, the total processing time, and some information about the accumulation process.

The SAS System

The TSMODEL Procedure

Processing Summary	
CAS table	PRICEDATA
Number of analysis variables	1
Number of rows read	2040
Number of groups read	17
Memory for group packages (KB)	2
Time to load groups (seconds)	0.92522192
Minimum time ID	JAN1998
Maximum time ID	DEC2002
Minimum time periods	60
Maximum time periods	60
Number of nodes run	121
Number of nodes with data	15
Number of nodes with groups	15
Number of threads budgeted	40
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	32
Maximum threads active	80
Number of groups processed by submitted code	17
Number of groups failing	0
Elapsed time to process groups (seconds)	1.7797710896
Number of array table rows produced	1020
Number of scalar table rows produced	17

Output 1. Summary for Processing Example 1

Output 2 shows a subset of the scalar output, which is produced from the following code:

```
PROC PRINT DATA=mycas.outscalar; RUN;
```

You can verify from Output 2 that all exit and return codes are 0.

Obs	regionName	productLine	productName	_STATUS_	runtime	exitCode	rc1	rc2	rc3	rc4	rc5	rc6
1	Region1	Line1	Product3	0	0.4798159599	0	0	0	0	0	0	0
2	Region2	Line2	Product7	0	0.4799671173	0	0	0	0	0	0	0
3	Region2	Line3	Product8	0	0.4737679958	0	0	0	0	0	0	0
4	Region3	Line5	Product16	0	0.4736320972	0	0	0	0	0	0	0
5	Region3	Line4	Product12	0	0.416203022	0	0	0	0	0	0	0
6	Region2	Line3	Product11	0	0.4436099529	0	0	0	0	0	0	0
7	Region3	Line4	Product15	0	0.4660289288	0	0	0	0	0	0	0
8	Region1	Line1	Product1	0	0.4725699425	0	0	0	0	0	0	0
9	Region3	Line4	Product13	0	0.7767958641	0	0	0	0	0	0	0
10	Region2	Line2	Product6	0	0.45221591	0	0	0	0	0	0	0

Output 2. Partial Output of OUTSCALAR Table Produced by Example 1

Output 3 shows the first 10 lines of the OUTARRAY table from this example. You can see the moving average values (MAVG) in the last column. Note that values are obtained at the

boundaries since the convolution mode was set to "same." This output is generated via the following command:

```
PROC PRINT DATA=mycas.outarray; RUN;
```

Obs	regionName	productLine	productName	_STATUS_	_SEASON_	_CYCLE_	date	sale	MAVG
1	Region1	Line1	Product3	0	1	1	JAN1998	300	200.33333333
2	Region1	Line1	Product3	0	2	2	FEB1998	301	320
3	Region1	Line1	Product3	0	3	3	MAR1998	359	352.33333333
4	Region1	Line1	Product3	0	4	4	APR1998	397	366.66666667
5	Region1	Line1	Product3	0	5	5	MAY1998	344	362
6	Region1	Line1	Product3	0	6	6	JUN1998	345	344.33333333
7	Region1	Line1	Product3	0	7	7	JUL1998	344	342.33333333
8	Region1	Line1	Product3	0	8	8	AUG1998	338	341.66666667
9	Region1	Line1	Product3	0	9	9	SEP1998	343	334.66666667
10	Region1	Line1	Product3	0	10	10	OCT1998	323	324.66666667

Output 3. Partial Output of OUTARRAY Table of Example 1

EXAMPLE 2: ARIMA FORECASTING USING R

This is a more realistic example, which demonstrates how to apply an ARIMA model implemented in R to your data. To keep the example succinct, the R model is used exclusively. However, the example can be extended to work with other objects to do things like include the custom model in the automated model selection process that is provided by the ATSM package objects; see *SAS Visual Forecasting: Time Series Packages*. This example also demonstrates how you can load source code from a file. The freely available forecast package[1] for R is required for this example.

As with the previous example, the first step is to establish a connection to the CAS server and create a CAS library called *mycas*. The *mycas* library enables you to transfer data sets to the CAS server where the distributed time series analysis is performed.

```
CAS mycas;
LIBNAME mycas CAS SESSREF = mycas;
```

A DATA step transfers the *Sashelp.PriceData* data set into the CAS *mycas* library:

```
DATA mycas.pricedata;
  SET sashelp.pricedata;
RUN;
```

The PROC TSMODEL statement specifies the input data set (*mycas.pricedata*), an output table in which to store the output data set (*mycas.outarray*), an output table in which to store one or more scalar variables (*mycas.outscalar*), and an output table in which to store two output objects (the object *mycas.rlog* stores all the output that is generated by the R program and the object *rvars* stores information about shared variables). LEAD=12 requests that 12 time steps into the future be forecasted.

```
PROC TSMODEL DATA=mycas.pricedata OUTARRAY=mycas.outarray
             OUTSCALAR=mycas.outscalar
             OUTOBJ=(rlog=mycas.rlog rvars=mycas.rvars)
             LEAD=12;
```

[1] https://cran.r-project.org/web/packages/forecast/forecast.pdf

The ID statement specifies the variable date as the time index variable, and the INTERVAL= option indicates that the data are monthly.

```
ID date INTERVAL = MONTH;
```

The BY statement specifies that each unique combination of the data set variables regionname, productline, and productname correspond to a unique time series BY group. BY groups are processed independently.

```
BY regionname productline productname;
```

The VAR statement specifies the input data set variable SALE. The ACCUMULATE=AVG option specifies an average value accumulation for the SALE variable.

```
VAR SALE / ACCUMULATE = AVG;
```

The OUTSCALAR statement specifies the scalar variables that the SAS script is to generate and store. These include variables in which to store the R program's run time (runtime), exit code (exitCode), and the return code from each method called for the R object (rc1–rc7):

```
OUTSCALAR runtime exitCode rc1 rc2 rc3 rc4 rc5 rc6 rc7;
```

The OUTARRAY statement specifies the array variables that the SAS script is to generate and store. The only output array is the series that is modeled using the R ARIMA model (rPred).

```
OUTARRAY rPred;
```

The REQUIRE statement specifies the EXTLANG package, which includes all the objects that are needed for SAS to interact with external languages.

```
REQUIRE EXTLANG;
```

The program statements between the SUBMIT and ENDSUBMIT statements use the EXTLANG package objects to run the R program on the CAS server:

```
SUBMIT;

    /*
     * Initialize the R object, which is the interface to the
     * R interpreter. The interpreter executable is set via a
     * CAS configuration file.
     */
    declare object robj(R) ;
    rc1 = robj.Initialize() ;

    /*
     * Push code from the filesystem. The R object will dynamically create
     * a file that contains all source code to be run and will autogenerate
     * code for transferring to and from the SAS environment.
     * The file r_arima_code.r has the following contents:
          ------------------------------------------------------
```

```
            library(forecast)
            Y<- Y[1:(NFOR - HORIZON)]
            Y_ts<-ts(Y,frequency=12)
            LOG_Y_ts<-log(Y_ts)
            fit <- stats::arima(LOG_Y_ts, order=c(p=0, d=1, q=1),
            seasonal=list(order=c(0,1,1), frequency=12))
            sse<-sum(fit$residuals^2)
            forecast(fit)
            a <- stats::predict(fit, n.ahead=HORIZON)
            PREDICT <- c( exp(fitted.values(fit)), exp(a$pred) )
            ----------------------------------------------------
         */
        rc2 = robj.PushCodeFile('/path/to/r_arima_code.r') ;

        /*
         * Specify variables to share between SAS and R.
         * SALE is the (READONLY) dependent variable. The ARIMA code
         * uses the generic name Y for the dependent variable, so
         * SALE is aliased to Y.
         * rPred will contain the predicted series, which is returned to the
         * SAS program. The R code that is used stores the predicted series in
         * the variable PREDICT, so rPred is aliased to PREDICT.
         * Two additional read-only variables are needed by the R code:
         * NFOR, which stores the forecast length, and HORIZON, which stores
         * the forecast horizon.
         */
        rc3 = robj.AddVariable(SALE, 'ALIAS', 'Y') ;
        rc4 = robj.AddVariable(rPred, 'ALIAS', 'PREDICT', 'READONLY', 'FALSE');
        rc5 = robj.AddVariable(_LENGTH_, 'ALIAS', 'NFOR') ;
        rc6 = robj.AddVariable(_LEAD_,'ALIAS','HORIZON') ;

         /*
          * Run the model and get the exit code and run time.
          */
        rc7 = robj.Run() ;
        exitCode = robj.GetExitCode() ;
        runtime = robj.GetRunTime() ;

        /*
         * Store the execution and resource usage statistics logs.
         */
        declare object rlog(OUTEXTLOG) ;
        rc16 = rlog.Collect(robj, 'EXECUTION') ;
        declare object rvars(OUTEXTVARSTATUS) ;
        rc17 = rvars.collect(robj) ;

    ENDSUBMIT;
    RUN;
```

The PROC TSMODEL summary is shown in Output 4.

The TSMODEL Procedure

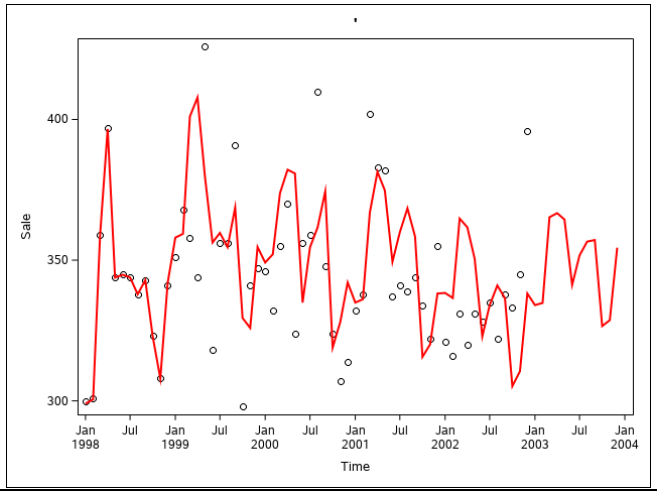
Processing Summary	
CAS table	PRICEDATA
Number of analysis variables	1
Number of rows read	2040
Number of groups read	17
Memory for group packages (KB)	2
Time to load groups (seconds)	5.2867841721
Minimum time ID	JAN1998
Maximum time ID	DEC2002
Minimum time periods	60
Maximum time periods	60
Number of nodes run	121
Number of nodes with data	15
Number of nodes with groups	15
Number of threads budgeted	40
Minimum thread group count	0
Maximum thread group count	1
Minimum threads active	32
Maximum threads active	80
Number of groups processed by submitted code	17
Number of groups failing	0
Elapsed time to process groups (seconds)	6.8440380096
Number of array table rows produced	1224
Number of scalar table rows produced	17

Output 4. PROC TSMODEL Summary Statistics for Example 2

The following code produces a plot that shows the actual and forecasted units sold over time for a simple BY group. The forecast values come from the R ARIMA model that is used in Example 2. The DATA step creates a subseries that consists of the BY group that pertains to Region 1, Product Line 1, and Product name "Product 3". The SGPLOT procedure uses the subseries that was obtained in the DATA step to create a scatter plot of the actual SALE values along with a line plot of the values that were obtained by the R ARIMA model. The output is shown in Output 5.

```
DATA mycas.subseries ;
    set mycas.outarray(where=(regionName="Region1" and productLine="Line1"
and productName="Product3")) ;
RUN ;

PROC SGPLOT data=mycas.subseries ;
   scatter x=DATE y=SALE / markerattrs=(color=black) LEGENDLABEL = 'Actual'
Name="act";
   series x=DATE y=rPRED / lineattrs=(color=red thickness=2)
                                LEGENDLABEL = 'Predicted' Name="pred";
   yaxis LABEL="Sale" ;
   xaxis LABEL="Time" ;
   keylegend / POSITION=BOTTOMRIGHT LOCATION=INSIDE ACROSS=1 ;
RUN;
```



Output 5. Actual and Modeled Sales for R-Based ARIMA Model

This R program is not expected to produce any text output if no errors occur, so _LOGTEXT_ should be empty. This can be verified by looking at the *mycas.rlog* table. The following code produces this table:

```
PROC PRINT DATA=mycas.rlog; RUN;
```

The output is shown in Output 6.

Obs	regionName	productLine	productName	_RUNID_	_EXITCODE_	_RUNTIMESECONDS_	_LOGTYPE_	_LOGLEN_	_LOGTEXT_
1	Region1	Line1	Product3	UNNAMED RUN	0	2.899353981	EXECUTION	0	
2	Region2	Line2	Product7	UNNAMED RUN	0	2.8814299107	EXECUTION	0	
3	Region2	Line3	Product8	UNNAMED RUN	0	2.9811868668	EXECUTION	0	
4	Region3	Line5	Product16	UNNAMED RUN	0	2.980383873	EXECUTION	0	
5	Region3	Line4	Product12	UNNAMED RUN	0	2.7156209946	EXECUTION	0	
6	Region2	Line3	Product11	UNNAMED RUN	0	2.7369279861	EXECUTION	0	
7	Region3	Line4	Product15	UNNAMED RUN	0	2.7010149956	EXECUTION	0	
8	Region1	Line1	Product1	UNNAMED RUN	0	2.7325601578	EXECUTION	0	
9	Region3	Line4	Product13	UNNAMED RUN	0	3.2494049072	EXECUTION	0	
10	Region2	Line2	Product6	UNNAMED RUN	0	2.7260670662	EXECUTION	0	
11	Region3	Line5	Product17	UNNAMED RUN	0	2.7100639343	EXECUTION	0	
12	Region2	Line3	Product10	UNNAMED RUN	0	2.7092959881	EXECUTION	0	
13	Region2	Line2	Product4	UNNAMED RUN	0	2.7104790211	EXECUTION	0	
14	Region1	Line1	Product2	UNNAMED RUN	0	4.3636190891	EXECUTION	0	
15	Region2	Line3	Product9	UNNAMED RUN	0	3.3587551117	EXECUTION	0	
16	Region2	Line2	Product5	UNNAMED RUN	0	2.7048690319	EXECUTION	0	
17	Region3	Line4	Product14	UNNAMED RUN	0	3.8954088688	EXECUTION	0	

Output 6. Output of RLOG Table for Example 2

EXAMPLE 3: LIGHTNING-FAST BIG DATA ANALYSIS

This example demonstrates the TSMODEL procedure's ability to perform fast time series analysis on big data. By using the EXTLANG package objects with PROC TSMODEL, any Python or R code can be seamlessly parallelized at a large scale, with minimal overhead. In this example, a large industrial data set that consists of more than 1.5 million BY groups is processed. On average, each BY group contains 4.3 years of weekly historical data. This example was conducted on a cluster of 128 worker nodes and 32 BY-group threads per node. Each worker node contains dual Intel Xeon E5-2680 CPUs; each CPU has 8 cores with Intel Hyper-Threading Technology. Each worker has 252 GB of RAM.

First, a connection to the CAS server is established, and a CAS library called *mycas* is created. The *mycas* library enables you to transfer data sets to the CAS server where the distributed time series analysis is performed:

```
CAS mycas;
LIBNAME mycas CAS SESSREF = mycas;
```

The DATA step is omitted from this example because the data are not publicly available. The PROC TSMODEL statement is similar to the previous examples. The details about the statements before the SUBMIT…END SUBMIT block are omitted for brevity.

```
PROC TSMODEL data = mycas.large_distributor
          OUTARRAY=mycas.outarray
          OUTSCALAR=mycas.outscalar
          LEAD=52
         ;
 ID  period_start_dt
     interval = week
     setmissing = missing
     trimid = none
 ;
 BY customer_id product_id store_location_id;
 VAR demand_qty / acc = total;
 OUTSCALAR pyTime pyExitCode pyProcessingTime
           prc1 prc2 prc3 prc4 prc5 prc6 prc7 prc8 prc9 prc10
           prc11 prc12 prc13 prc14 prc15 prc16
           ;
 OUTARRAY pyPred ;
 REQUIRE extlang ;
```

The code within the SUBMIT…ENDSUBMIT block runs on all the BY groups. The Python version 3 interpreter that is specified in the CAS configuration is launched once for each of the 32 worker threads. The interpreter process is duplicated for each new BY group that the thread processes. Python modules that are used in the user program are preloaded in the duplicated interpreter process to further reduce overhead. Note that Python's indention rules must be obeyed.

```
SUBMIT;

 declare object py(PYTHON3);
 prc1=py.Initialize();

 /* Create the script */
 prc1 = py.PushCodeLine('import numpy as np');
 prc2 = py.PushCodeLine("import os") ;
 prc3 = py.PushCodeLine("import time") ;
 prc4 = py.PushCodeLine('start = time.time()');
 prc5 = py.PushCodeLine('try:');
 /* Moving average with window size = 7 */
 prc6 = py.PushCodeLine('   w = np.ones((7,))/7 ; ');
 prc7 = py.PushCodeLine('   fit = np.convolve(Y, w, mode="same")') ;
 prc8 = py.PushCodeLine('   PREDICT = fit') ;
 prc10 = py.PushCodeLine('except Exception as e:');
 prc11 = py.PushCodeLine('   print("Error occured during computation.
                             Y values: {0}. Error: {1}".format(Y, e))') ;
 prc12 = py.PushCodeLine('PYPROCESSINGTIME = time.time() - start') ;

 /* Add variables */
 prc13 = py.AddVariable(demand_qty, 'ALIAS', 'Y') ;
```

```
 prc14 = py.AddVariable(pyPred, 'ALIAS', 'PREDICT', 'READONLY', 'FALSE');
 prc15 = PY.AddVariable(pyProcessingTime, 'READONLY', 'FALSE');

 /* Run the program */
 prc16 = py.Run();
 pyTime = py.GetRuntime();
 pyExitCode = py.GetExitCode() ;
ENDSUBMIT;
RUN;
```

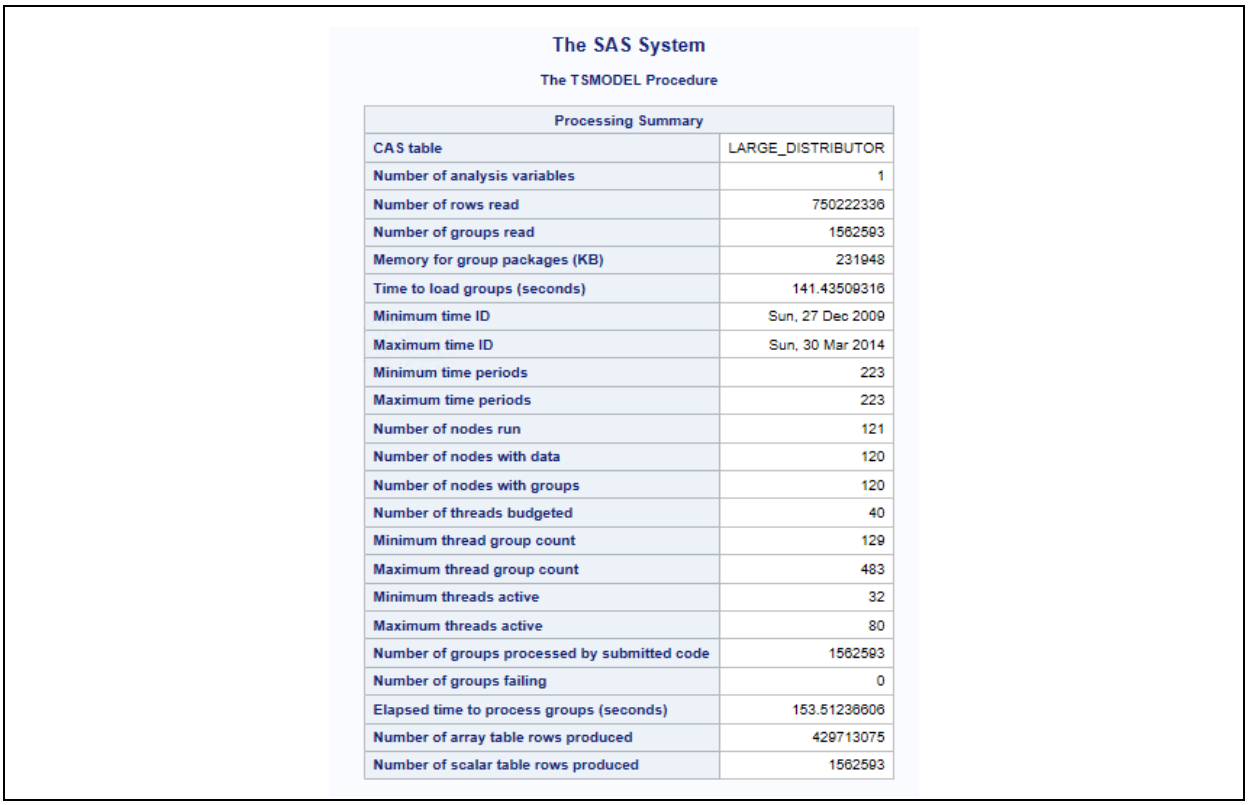
The results for this example are split into two parts. The first part contains results that are obtained by running without any Python code. These results assess the overhead of just *shuffling* the data among the worker nodes. The second part runs the preceding code. Hence, the second part of the results shows the additional overhead of loading the Python interpreter and transferring data between the interpreter and SAS, in addition to shuffling the data. The output summary of the first part is shown in Output 7, which shows that 1,562,593 BY groups were processed in 29.1 seconds. The output summary from the full example is shown in Output 8, which shows that processing the same BY groups took 153.5 seconds. Given the simplicity of this program, most of this time can be attributed to the overhead involved in duplicating the Python process for each BY group and loading and storing their data. Hence, the penalty for processing 750 million rows of data distributed among 1.5 million BY groups was just 124 seconds, which is quite small.

The SAS System

The TSMODEL Procedure

Processing Summary	
CAS table	LARGE_DISTRIBUTOR
Number of analysis variables	1
Number of rows read	750222336
Number of groups read	1562593
Memory for group packages (KB)	231948
Time to load groups (seconds)	20.797576904
Minimum time ID	Sun, 27 Dec 2009
Maximum time ID	Sun, 30 Mar 2014
Minimum time periods	223
Maximum time periods	223
Number of nodes run	121
Number of nodes with data	120
Number of nodes with groups	120
Number of threads budgeted	40
Minimum thread group count	129
Maximum thread group count	483
Minimum threads active	32
Maximum threads active	80
Number of groups processed by submitted code	1562593
Number of groups failing	0
Elapsed time to process groups (seconds)	29.101961136
Number of array table rows produced	429713075
Number of scalar table rows produced	1562593

Output 7. PROC TSMODEL Summary for Loading Large Data Set Using 121 Workers



The SAS System

The TSMODEL Procedure

Processing Summary	
CAS table	LARGE_DISTRIBUTOR
Number of analysis variables	1
Number of rows read	750222336
Number of groups read	1562593
Memory for group packages (KB)	231948
Time to load groups (seconds)	141.43509316
Minimum time ID	Sun, 27 Dec 2009
Maximum time ID	Sun, 30 Mar 2014
Minimum time periods	223
Maximum time periods	223
Number of nodes run	121
Number of nodes with data	120
Number of nodes with groups	120
Number of threads budgeted	40
Minimum thread group count	129
Maximum thread group count	483
Minimum threads active	32
Maximum threads active	80
Number of groups processed by submitted code	1562593
Number of groups failing	0
Elapsed time to process groups (seconds)	153.51236606
Number of array table rows produced	429713075
Number of scalar table rows produced	1562593

Output 8. PROC TSMODEL Summary for Example 3

CONCLUSION

The TSMODEL procedure enables both scalable and optimized time series analysis in a cloud environment. PROC TSMODEL comes equipped with a generic scripting environment, which enables you to develop custom time series analysis algorithms and prepare your data for analysis (clean, transform, preprocess, and postprocess), all within the same script. This environment helps reduce data movement (because the data remain in the same contiguous memory throughout the analysis) and also optimizes code development. PROC TSMODEL also comes equipped with various specialized time series analysis packages that provide advanced support for time series analysis (in the time domain or in the frequency domain), time series decomposition, time series modeling, signal analysis and anomaly detection (for IoT), and temporal data mining. External language support extends the SAS scripting environment to allow for open-source integration. You can integrate new and existing Python and R programs into your SAS script in order to enhance your processing or analysis (or both). Because of these features, what can be accomplished by PROC TSMODEL is limited only by your imagination. As is illustrated by the third example, PROC TSMODEL's distributed nature (processing BY groups in parallel), efficiency and scalability (minimizing I/O, performing all data operations in memory, and reusing allocated memory efficiently across BY groups), and optimized time series modeling and forecasting capabilities enable big data forecasting problems to be solved with unprecedented speeds. In summary, TSMODEL can efficiently perform time series analysis of big data in close to real time.

REFERENCES

Quirino, T., Leonard, M., and Blair, E. 2018. "Scalable Cloud-Based Time Series Analysis and Forecasting." *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc.

Available http://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2020/2020-4440.pdf

ACKNOWLEDGMENTS

The authors would like to thank Anne Baxter from SAS for her editing and Ed Blair for his technical insight.

CONTACT INFORMATION

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

Javier Delgado
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
919-531-4385
Javier.Delgado@sas.com

Thiago Quirino
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
919-531-3721
Thiago.Quirino@sas.com

Michael Leonard
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
919-531-6967
Michael.Leonard@sas.com