Proc DS2

daniel@swe.sas.com

Agenda

- What is and isn’t DS2?
- Introduction to the language
- Comparison of Data Step code and DS2Code
- VDMML – a use case
- Demo

Introduction to the Language

DS2 Structure

Follows a structured-programming and object-oriented approach

- Begins with Proc DS2
- Consists of a combination of
  - Declare statements to define variables
  - Pre-defined methods
  - User-defined methods
  - The basic Methods…
    - Init → runs once
    - Run → multiple times
    - Term → at end of processing

Variable Types

PROC DS2 provides an extended list of data SAS® data types which allows for seamless interaction among SAS® and external databases.

- BIGINT, BIGINT (INT64)
- BINARY(n)
- DATE
- DECIMAL|NUMERIC(p,s)
- FLOAT
- NCHAR(n)
- NVARCHAR(n)
- REAL
- VARBINARY(n)
- INTEGER, INTEGER (INT32)
- CHAR, CHAR(n)
- DOUBLE
- VARCHAR, VARCHAR(n)
- SMALLINT
- TINYINT
- TIME(p)
- TIMESTAMP(p)

DS2 Method blocks

Pre-defined

Block	Identifiers	Description	Scope
Data Program	DATA …… ENDDATA	Basic method block for processing data	Variables declared at top of block and in a SET statement are global and included in the PDV Global scope last for duration of program
Package	PACKAGE ….. ENDPACKAGE	Combination of other data and method blocks to address a specific requirement	Variables declared at top of block and in a SET statement are global , they are not included in the PDV Global scope last for duration of package
Thread Program	THREAD …… ENDTHREAD	Allows multiple processes to run in parallel	Variables declared at top of block and in a SET statement are global , unless explicitly dropped they are passed to the output Global scope last for duration of package, but can be passed to the Data Program using the SET statement
Method	METHOD ….. ENDMETHOD	Allows the user to create their own set of code for reuse	Sub block used within one of the above blocks, method name global in calling block, variable declared within block , local and only last for duration of method
Do Loop	DO ….. END		Not applicable

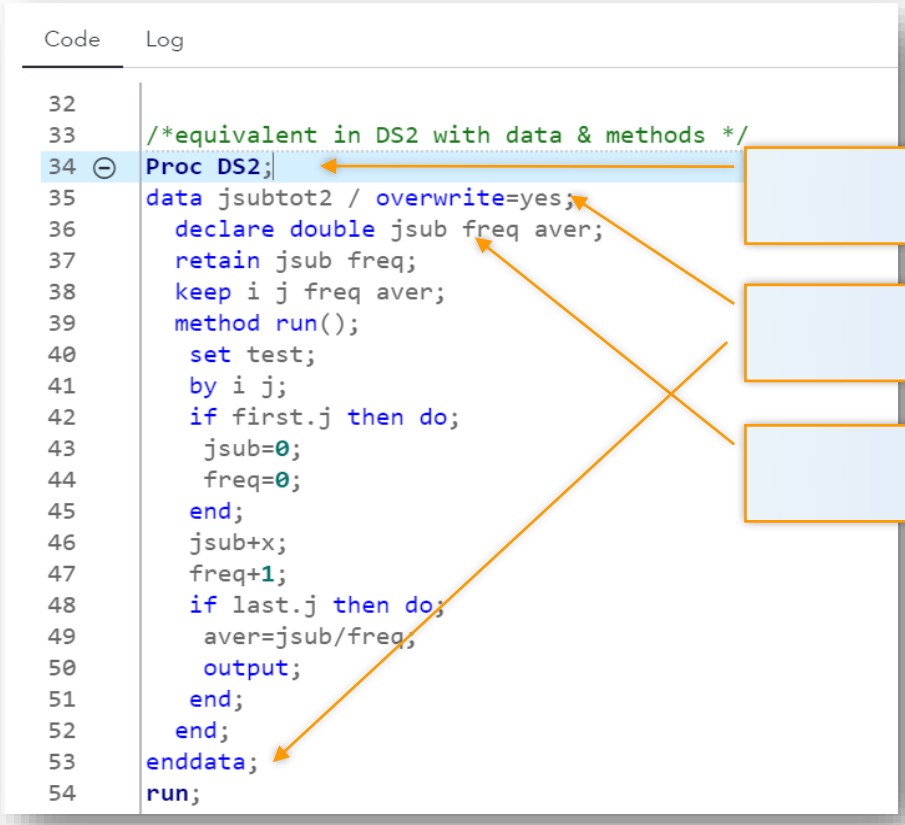
Data Step versus DS2

Key highlights

Data step	DS2
Code resides in Data step or proc	Code resides in Methods blocks
Use Proc SQL to access data, 1 database per Proc	FedSQL embedded in Proc DS2 allowing access to multiple databases at the same time
No concept of variable scope	Variables can be local or global
Character & numeric variables	Ansi SQL data types – leading to better precision in calculations and fixed and variable length characters

Full comparison between data step and DS2 code can be found here

Initial Concepts



Proc DS2 to start the process

Data Block being used to process data

Variables being declared

Initial Concepts

Code Log

```
32
33  /*equivalent in DS2 with data & methods */
34  Proc DS2;
35  data jsubtot2 / overwrite=yes;
36    declare double jsub freq aver;
37    retain jsub freq;
38    keep i j freq aver;
39    method run();
40     set test;
41     by i j;
42     if first.j then do;
43      jsub=0;
44      freq=0;
45     end;
46     jsub+x;
47     freq+1;
48     if last.j then do;
49      aver=jsub/freq;
50      output;
51     end;
52    end;
53  enddata;
54  run;
```

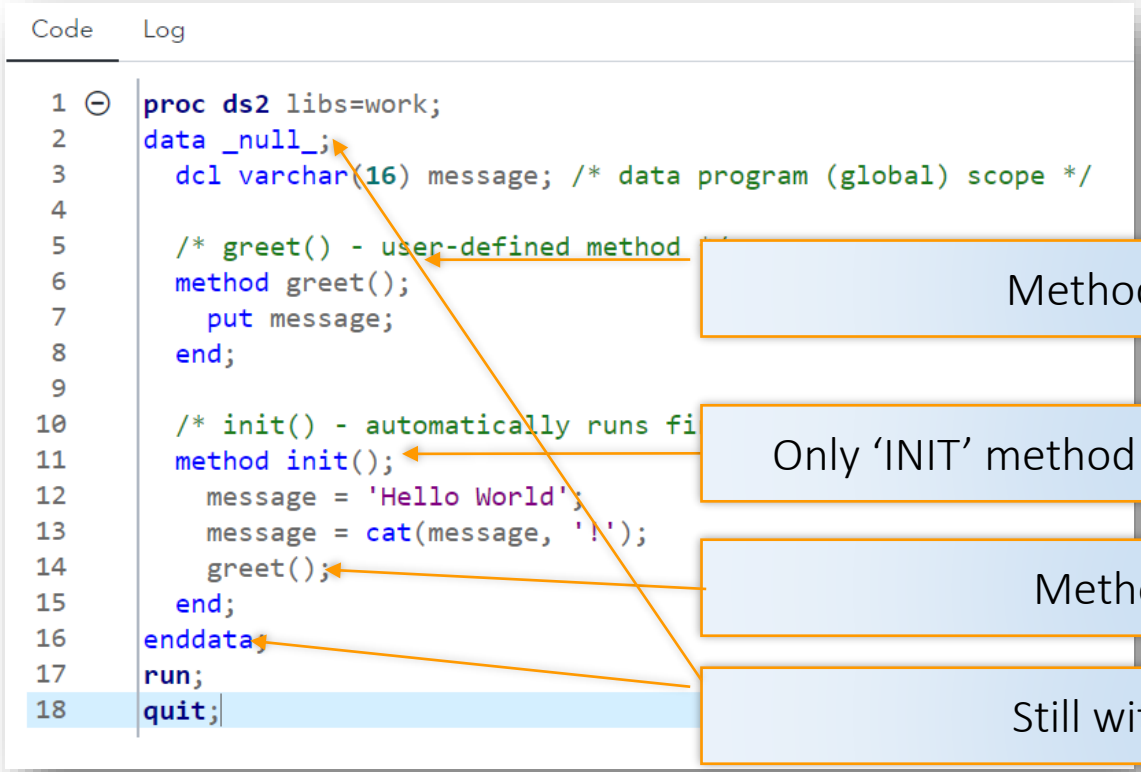
‘Run’ method being used to process over each row

No explicit ‘INIT’ or ‘TERM’ methods used

Method overview

- DS2 methods are like functions, procedures, subroutines, and the methods of object-oriented languages such as Java.
- A *method* can be thought of as a module that contains a sequence of instructions to perform a specific task, enables you to break up a complex problem into smaller modules.
- 2 types of Methods:
  - System methods
    - Init → runs once a start of program
    - Run → runs for each execution of the program
    - Term → runs at the end of the program
    - SetParms → used in a data program to initialise values of parameters for a thread
    - Not all methods have to be explicitly defined
  - User-defined methods

User defined methods



Method ‘Greet’ defined

Only ‘INIT’ method used, single pass of program

Method ‘Greet’ used

Still within a data block

Package Block

Is a collection of methods and variables

- USER DEFINED PACKAGES
  - These are packages that you can use to store methods for any purpose.
  - These packages can be thought of as libraries of your methods. Any type of method can be saved in a package.
  - Use the PACKAGE statement to store methods in a package
- PREDEFINED PACKAGES
  - SAS provides some predefined packages in DS2.
- To use a package, a DS2 program, another package, or a thread instantiates the package and accesses its method
  - Using the a DECLARE statement or with the _NEW_ operator

Pre Defined Packages

- FCMP – allows you to create your own functions for reuse
- SQLSTMT –FEDSQL access to external databases, not supported on CAS
- MATRIX - provides a powerful and flexible matrix programming capability not supported on CAS
- JSON - interface to create and parse JSON text
- Logger - interface with the SAS logging facility
- HTTP – constructs an HTTP client in order to access HTTP web servers
- Hash - efficient, convenient mechanism for quick data storage and retrieval
- Hash iterator
- PCRXFIND and PCRXREPLACE - provides a way to find a substring within a given string and replace a substring not supported on CAS
- TZ – process local & international data time values

Package example

Code used to allow Python code to be called from within Viya

score.sas

```
1  package pythonScore / overwrite=yes;
2  dcl package pymas pm;
3  dcl package logger logr('App.tk.MAS');
4  dcl varchar(32767) character set utf8 pyp
5  dcl int resultCode revision;
6
7  method score(double LOAN, double MORTDUE, double VALUE, double YOJ, double DEROG, double DELINQ, double CLAGE, double N
8     resultCode = revision = 0;
9     if null(pm) then do;
10       pm = _new_ pymas();
11       resultCode = pm.useModule('model_ex
12       if resultCode then do;
13          resultCode = pm.appendSrcLine('import numpy');
14          resultCode = pm.appendSrcLine('import pandas');
15          resultCode = pm.appendSrcLine('import pickle');
16          resultCode = pm.appendSrcLine('');
17          resultCode = pm.appendSrcLine('#inside input parameter goes all the predictor/input variables');
18          resultCode = pm.appendSrcLine('def scoreFleetDTModel (LOAN,MORTDUE,VALUE,YOJ,DEROG,DELINQ,CLAGE,NINQ,CLNO,DEBT
19          resultCode = pm.appendSrcLine('    "Output: EM_EVENTPROBABILITY, EM_CLASSIFICATION"');
20          resultCode = pm.appendSrcLine('');
21          resultCode = pm.appendSrcLine('    try:');
22          resultCode = pm.appendSrcLine('        _thisModelFit');
23          resultCode = pm.appendSrcLine('    except NameError:');
```

Main package defined and called pythonScore

Uses Pymas package user-defined, but provided with MM

Also uses Logger pre-defined package & instantiates it

Package example

Code used to allow Python code to called from within SAS

score.sas

```
1  package pythonScore / overwrite=yes;
2  dcl package pymas pm;
3  dcl package logger logr('App.tk.MAS');
4  dcl varchar(32767) character set utf8 pyPgm;
5  dcl int resultCode revision;
6
7  method score(double LOAN, double MORTDUE, double VALUE, double YOJ, double DEROG, double DELINQ, double CLAGE, double N
8     resultCode = revision = 0;
9     if null(pm) then do;
10       pm = _new_ pymas();
11       resultCode = pm.useModule('model_exec_731ba
12       if resultCode then do;
13          resultCode = pm.appendSrcLine('import numpy');
14          resultCode = pm.appendSrcLine('import pandas');
15          resultCode = pm.appendSrcLine('import pickle')
16          resultCode = pm.appendSrcLine('');
17          resultCode = pm.appendSrcLine('#inside input p
18          resultCode = pm.appendSrcLine('def scoreFleetD
19          resultCode = pm.appendSrcLine('    "Output: EM
20          resultCode = pm.appendSrcLine('');
21          resultCode = pm.appendSrcLine('    try:');
22          resultCode = pm.appendSrcLine('        _thisMo
23          resultCode = pm.appendSrcLine('    except Name
```

Method score defined in the package

Instantiates the use of pymas package

Uses dot notation to leverage the methods within the package…

→ useModule

→ appendSrcLine..

Pre Defined Packages

- FCMP – allows you to create your own functions for reuse
- SQLSTMT –FEDSQL access to external databases, not supported on CAS
- MATRIX - provides a powerful and flexible matrix programming capability not supported on CAS
- JSON - interface to create and parse JSON text
- Logger - interface with the SAS logging facility
- HTTP – constructs an HTTP client in order to access HTTP web servers
- Hash - efficient, convenient mechanism for quick data storage and retrieval
- Hash iterator
- PCRXFIND and PCRXREPLACE - provides a way to find a substring within a given string and replace a substring not supported on CAS
- TZ – process local & international data time values

SQLSTMT Package

- Passes FedSQL statements to a DBMS for execution and accesses the result set returned by the DBMS.
- DS2 would allow the use of data step code and FEDSQL code to be used in one pass of the data. for example
  - SQL statement – create, joining
  - Data step to process afterwards

SQLSTMT Example

Package definition using SQLSTMT package within

```
package fibonnaci;
   declare int nMax;    /* maximum index in this series */

   method fibonnaci(int n);
      nMax = n;
   end;

   method output(char(100) tablename);
      declare int    n;    /* index of fib in Fibonacci series */
      declare double fib;  /* Fibonacci number */

      declare package sqlstmt stmt;
      /* note that stmt is not created */

      sqlexec('create table ' || tablename || '
         (n int, fib double)');

      /* Using _new_ allows delay of prepare of SQL stat
      /* output table is created */

      stmt = _new_ sqlstmt('insert into ' || tablename |
         values (?, ?)');

      /* fibonnaci(0) = 0 */
      stmt.setInteger(1, 0);    /* column n */
      stmt.setDouble(2, 0);     /* column fib */
      stmt.execute();

      /* fibonnaci(1) = 1 */
      stmt.setInteger(1, 1);    /* column n */
      stmt.setDouble(2, 2);     /* column fib */
      stmt.execute();
```

SQLSTMT package declared

SQLSTMT package instantiated using _NEW_

Dot notation used to access package methods and populate data load

Thread Block

- Typically, DS2 code runs sequentially
- It is possible to run more than one process concurrently, using **threaded processing**. In threaded processing, each concurrently executing section of code is said to be running in a **thread**. DS2 threading works well both on a machine with multiple cores and within a massively parallel processing (MPP) database.
- A DS2 program processes input data and produces output data. A DS2 program can run in two different ways: as a program and as a thread.

Initial Concepts

Our original program based on a data block

Code Log

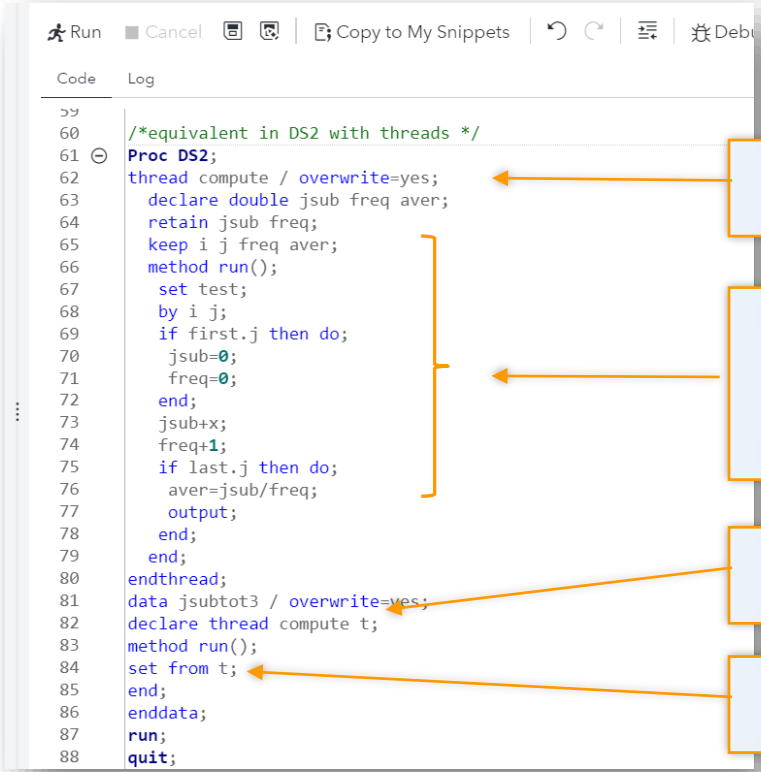
```
32
33  /*equivalent in DS2 with data & methods */
34  Proc DS2;
35  data jsubtot2 / overwrite=yes;
36    declare double jsub freq aver;
37    retain jsub freq;
38    keep i j freq aver;
39    method run();
40     set test;
41     by i j;
42     if first.j then do;
43      jsub=0;
44      freq=0;
45     end;
46     jsub+x;
47     freq+1;
48     if last.j then do;
49      aver=jsub/freq;
50      output;
51     end;
52    end;
53  enddata;
54  run;
```

Proc DS2 to start the process

Data Block being used to process data

Variables being declared

Data processing using threads



Thread block defined

This block of code is identical to the original DS2 data block program.

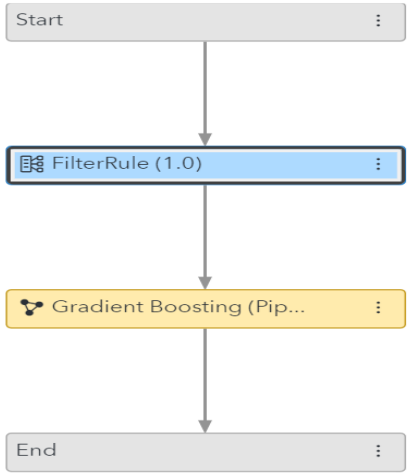
Thread declared and instantiated

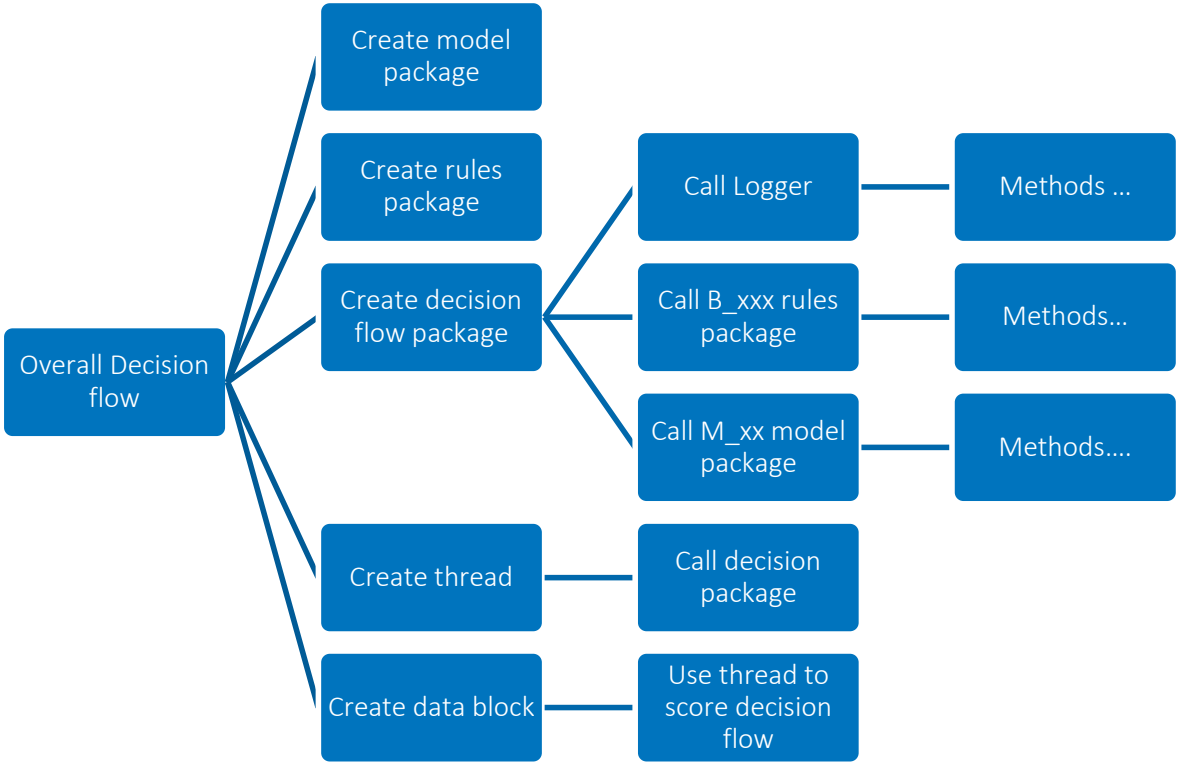
Thread called for each row of the table

Example – Use case

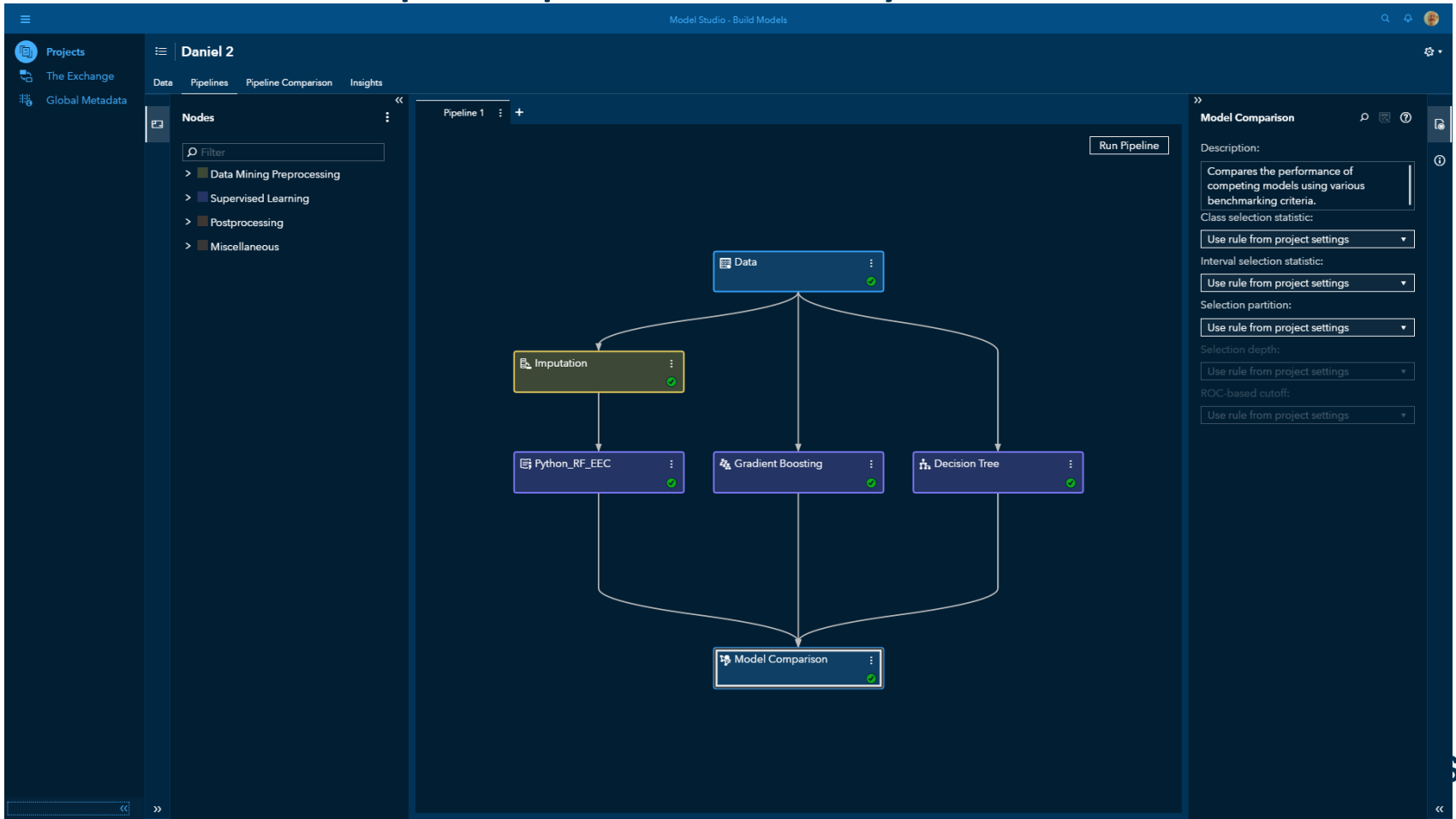
SAS Intelligent Decisioning uses DS2

Elements behind an ID decision flow for testing

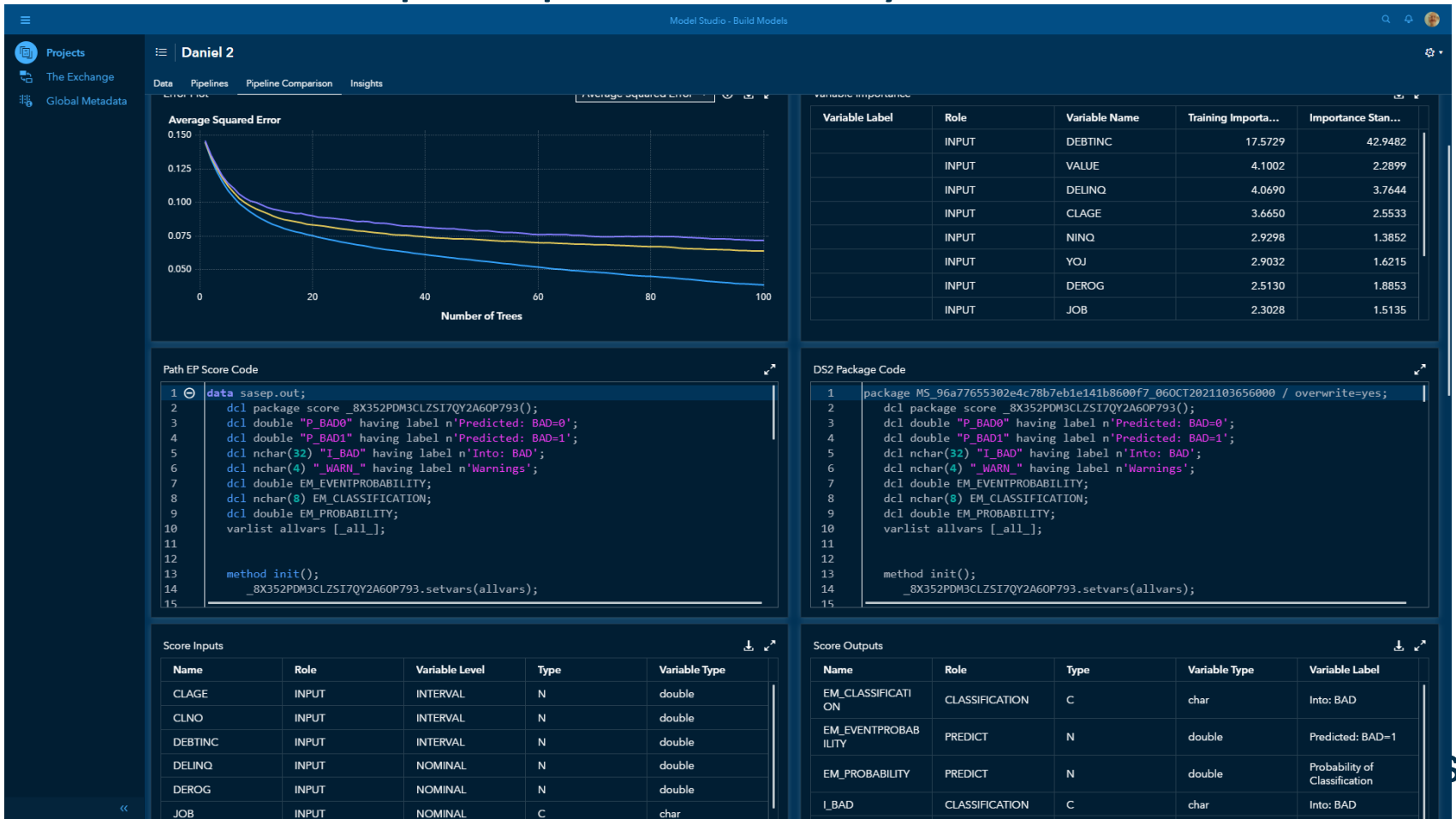




Simple Pipeline with Py and SAS



Simple Pipeline with Py and SAS



Simple Pipeline with Py and SAS

```
data sasep.out;
   dcl package score _8X352PDM3CLZSI7QY2A6OP793();
   dcl double "P_BAD0" having label n'Predicted: BAD=0';
   dcl double "P_BAD1" having label n'Predicted: BAD=1';
   dcl nchar(32) "I_BAD" having label n'Into: BAD';
   dcl nchar(4) "_WARN_" having label n'Warnings';
   dcl double EM_EVENTPROBABILITY;
   dcl nchar(8) EM_CLASSIFICATION;
   dcl double EM_PROBABILITY;
   varlist allvars [_all_];


   method init();
      _8X352PDM3CLZSI7QY2A6OP793.setvars(allvars);
      _8X352PDM3CLZSI7QY2A6OP793.setkey(n'EB3EB2011CC7520C1EB9E8772DB60149D1463B28');
   end;

   method post_8X352PDM3CLZSI7QY2A6OP793();
      dcl double _P_;

      if "P_BAD0" = . then "P_BAD0" = 0.8005033557;
      if "P_BAD1" = . then "P_BAD1" = 0.1994966443;
      if MISSING("I_BAD") then do ;
      _P_ = 0.0;
      if "P_BAD1" > _P_ then do ;
      _P_ = "P_BAD1";
      "I_BAD" = '1';
      end;
      if "P_BAD0" > _P_ then do ;
      _P_ = "P_BAD0";
      "I_BAD" = '0';
      end;
      end;
      EM_EVENTPROBABILITY = "P_BAD1";
      EM_CLASSIFICATION = "I_BAD";
      EM_PROBABILITY = MAX("P_BAD1", "P_BAD0");

   end;
```

```
package MS_96a77655302e4c78b7eb1e141b8600f7_06OCT2021103656000 / overwrite=yes;
   dcl package score _8X352PDM3CLZSI7QY2A6OP793();
   dcl double "P_BAD0" having label n'Predicted: BAD=0';
   dcl double "P_BAD1" having label n'Predicted: BAD=1';
   dcl nchar(32) "I_BAD" having label n'Into: BAD';
   dcl nchar(4) "_WARN_" having label n'Warnings';
   dcl double EM_EVENTPROBABILITY;
   dcl nchar(8) EM_CLASSIFICATION;
   dcl double EM_PROBABILITY;
   varlist allvars [_all_];


   method init();
      _8X352PDM3CLZSI7QY2A6OP793.setvars(allvars);
      _8X352PDM3CLZSI7QY2A6OP793.setkey(n'EB3EB2011CC7520C1EB9E8772DB60149D1463B28');
      _8X352PDM3CLZSI7QY2A6OP793.setOption('_destroyFirstInstLast_', 0);
   end;

   method post_8X352PDM3CLZSI7QY2A6OP793();
      dcl double _P_;

      if "P_BAD0" = . then "P_BAD0" = 0.8005033557;
      if "P_BAD1" = . then "P_BAD1" = 0.1994966443;
      if MISSING("I_BAD") then do ;
      _P_ = 0.0;
      if "P_BAD1" > _P_ then do ;
      _P_ = "P_BAD1";
      "I_BAD" = '1';
      end;
      if "P_BAD0" > _P_ then do ;
      _P_ = "P_BAD0";
      "I_BAD" = '0';
      end;
      end;
      EM_EVENTPROBABILITY = "P_BAD1";
      EM_CLASSIFICATION = "I_BAD";
      EM_PROBABILITY = MAX("P_BAD1", "P_BAD0");

   end;
```

Doc and links

- Full comparison between data step and DS2 code can be found here
- Proc DS2 reference
- Proc DS2 Programmer's Guide
- Getting Started with PROC DS2 – LexJansen
- The %TSLIT macro
- High-Performance Data Manipulation with SAS DS2 (Mark Jordan)

Demo

Q&A

sas.com