SESUG 2020 Paper 101

Getting Started with SAS® Viya and the R SWAT Package

Brian Varney, Experis Solutions

ABSTRACT

SAS Viya® gives SAS® and R developers the ability to collaborate and work off the same data sitting in memory on a SAS Viya server. This paper intends to show how to set up the connection from RStudio and process data using SAS Cloud Analytic Services (CAS). Examples will also be shown comparing the execution of analytics using R Studio and SAS Viya.

INTRODUCTION

SWAT stands for SAS Wrapper for Analytics Transfer. This package enables you to connect from **R** to a SAS Cloud Analytic Services host, run actions on in-memory tables, and work with the results of the actions.

The purpose of this paper is to help SAS Viya users that have a basic understanding of R get started in using this technology.

WHAT IS NEEDED TO GET STARTED WITH THE R SWAT PACKAGE?

The following is needed to get started using the R SWAT Package with SAS Viya:

1) A 64 bit SAS Viya programming-only environment either in Windows or Linux. A full SAS Viya deployment will also work. You must be on SAS Viya 3.1 or later.
2) A 64 bit R environment either on Linux or Windows. I would highly recommend using RStudio® as the development environment.
3) The ability for the SAS Viya and R environment servers to communicate with each other. Specifically, the R environment will need to be able to authenticate into the SAS Viya environment.
4) The R SWAT Package downloaded, installed, loaded in the R environment.
5) The dplyr, jsonlite, and httr packages installed and loaded in the R environment.
6) A basic understanding of coding in R.

GETTING SAS VIYA READY TO ACCEPT A CONNECTION

An active CAS session must be run on the SAS Viya server for R to connect to. Included below are a few lines of code that will start up a cas session for R to connect to.

```
/* initiate a CAS server connection listening for connection requests */
options cashost="localhost" casport=5570;

cas;

cas mySession sessopts=(caslib=casuser timeout=1800 locale="en_US");
```

METHODS TO CONNECT TO SAS VIYA USING THE R SWAT PACKAGE

As far as authentication goes, it is recommended to set up an authinfo file. Otherwise, you will need to supply your credentials in your connection code.

AUTHINFO FILE DOCUMENTATION

https://documentation.sas.com/?docsetId=authinfo&docsetTarget=n0xo6z7e98y63dn1fj0g9l2j7oyq.htm&docsetVersion=9.4&locale=en#n1stv9zynsyf6rn1wbr3ejga6ozf

There are two different methods for connecting to SAS Viya: binary and rest communication.

BINARY COMMUNICATION

This can only be used if your R environment is in Linux. An excerpt from the SAS documentation below shows advantages and disadvantages of using binary communication.

An example of a binary connection is:

conn_binary <- CAS("cloud.example.com", 5570)

There are no credentials in the connection definition above so an authinfo file is assumed to be set up. The following is an example of a binary connection with the credentials embedded.

conn_binary <- CAS('localhost', 5570, username="<username", password="password")

Advantages	Disadvantages
■ Communication is fast and efficient. There are fewer conversions between data types. ■ Data message handlers can be implemented to perform custom data loading. ■ Automatic encryption of communication if CAS is configured to perform TLS. ■ The SAS Threaded Kernel subsystem adds support for SAS data formats.	■ Platform support is limited because the SAS Threaded Kernel subsystem is a requirement. ■ The download and installation size is larger due to the addition of the SAS Threaded Kernel subsystem.

REST COMMUNICATION

This will work if your R environment is in Linux or Windows. An excerpt from the SAS documentation below shows advantages and disadvantages of using rest communication.

Following is an example of a rest connection:

```
conn_binary <- CAS('localhost', 8777, protocol='http')
```

There are no credentials in the connection definition so an authinfo file is assumed to be set up. The following is an example of a rest connection with the credentials embedded.

```
conn_rest <- CAS('localhost', 8777, protocol='http',
                          username="<username>",password="password")
```

Advantages	Disadvantages
■ Connections use standard HTTP and HTTPS communication. ■ The package uses R code only. It can be used on any platform that is supported by R. ■ The download and installation are smaller because the C libraries and SAS Threaded Kernel subsystem are not installed.	■ The conversion of objects to and from JSON is slower than binary. ■ HTTP is a less efficient communication protocol than binary. ■ Data message handlers for custom data loaders are not supported. ■ Extra data formatting features are not available, unless SAS Threaded Kernel is also installed.

The rest of the examples in this paper will leverage the binary connection. Submit the binary connection line of code below in R. The code and console results are shown below.

```
# R Code to connect to SAS Viya CAS
conn_binary <- CAS('localhost', 8777, protocol='http')
conn_binary
```

```
R Console excerpt:
NOTE: Connecting to CAS and generating CAS action functions for loaded
      action sets...
NOTE: To generate the functions with signatures (for tab completion), set
      options(cas.gen.function.sig=TRUE).
conn_binary
CAS(hostname=localhost, port=5570, username=bvarney, session=44f86478-d497-bd
49-bc21-2a7782ee2494, protocol=cas)
```

UNDERSTANDING CAS, CASL. ACTIONS, AND ACTION SETS

To effectively write code, we must first have a basic understanding of the mechanisms necessary to interact with SAS Viya from R. A brief explanation of the components follow:

CAS stands for Cloud Analytics Services. It is a cloud-based run-time environment for data management and analytics in SAS Viya.

CASL stands for Cloud Analytics Services Language. This is a language that can be used by SAS via PROC CAS or by other clients that can interact with CAS such as R, Python, & Lua. CASL is used to run code in CAS.

Actions are single tasks in CAS.

Action Sets are actions that are grouped together based on common functionality.

When we use the SAS R SWAT Package, it allows us to run processes in the SAS Viya CAS server from R using the functions from the action sets.

Many function names in CASL are typically constructed using the convention cas.<action set>.<action>().

For example, there is an action set for “Tables”. It contains actions such as “recordCount”. It would be called from R using syntax such as:

```
cas.table.recordCount(conn_binary, table='HMEQ')
```

The above code would return the following result to the RStudio console.

```
$RecordCount
     N
1 5960
```

The documentation for the action sets can be found at:

SAS® VIYA® 3.5 ACTIONS AND ACTION SETS BY NAME AND PRODUCT

https://documentation.sas.com/?cdcId=pgmcdc&cdcVersion=8.11&docsetId=allprodsactions&docsetTarget=titlepage.htm&locale=en

SHARING DATA

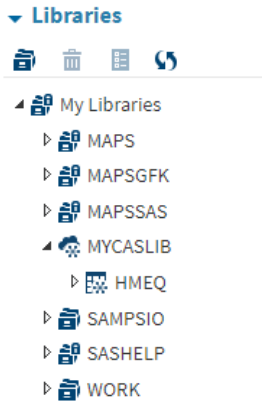
The connected R session can access the data on the SAS Viya server's CAS session. The following SAS Viya code loads a SAS Data Set into the SAS Viya server's memory. The promote option is important to allow the R session to be able to access the data.

```
libname mycaslib cas caslib=casuser;
proc casutil;
  load data=sampsio.hmeq casout="hmeq" outcaslib=casuser promote;
run;
```

SAS Viya Log Excerpt

```
72         libname mycaslib cas caslib=casuser;
 NOTE: Libref MYCASLIB was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 2d7ff5c6-82a0-f04c-9b75-2187e4ca23f9
73          proc casutil;
  NOTE: The UUID '2d7ff5c6-82a0-f04c-9b75-2187e4ca23f9' is connected using
session MYSESSION.
 74             load data=sampsio.hmeq casout="hmeq" outcaslib=casuser
promote;
NOTE: SAMPSIO.HMEQ was successfully added to the "CASUSER" caslib as "hmeq".
 75           run;
```

The HMEQ Data Set in the MYCASLIB SAS Library



ACCESSING THE SAS VIYA DATA FROM R

Now that the HMEQ data is sitting in the memory of the SAS Viya server and we have already established the connection from R, we can run a cas.table.tableinfo() function to list the contents of the SAS CAS library. The output is wrapped but you should be able to see the HMEQ table in there with 5,960 rows, 13 columns, etc.

Showing Available Data Sets in SAS Viya CAS Library

```
cas.table.tableInfo(connection_binary)
$TableInfo
  Name Rows Columns IndexedColumns Encoding       CreateTimeFormatted          ModTim
eFormatted       AccessTimeFormatted
1 HMEQ 5960      13              0    utf-8 2020-09-22T16:48:09-04:00 2020-09-22T16:4
8:09-04:00 2020-09-22T16:59:56-04:00
  JavaCharSet CreateTime    ModTime AccessTime Global Repeated View SourceName Source
Caslib Compressed Creator Modifier
1        UTF8 1916426889 1916426889 1916427596      1        0    0
0 bvarney
  SourceModTimeFormatted SourceModTime
1                                  NaN
```

Using HMEQ from the SAS Viya CAS Library

The command below sets up an R Object that is a pointer and can be accessed as you would an R data frame for some R functions.

```
hmeq_fromcas <- defCasTable(conn_binary, "HMEQ")
```

Name	Type	Value
hmeq_fromcas	S4 [5960 x 13] (swat::CASTab	S4 object of class CASTable
conn	S4 [1] (swat::CAS)	
tname	character [1]	'HMEQ'
caslib	character [1]	''
where	character [1]	''
orderby	list [0]	List of length 0
groupby	list [0]	List of length 0
gbmode	character [1]	''
computedOnDemand	logical [1]	FALSE
computedVars	character [1]	''
computedVarsProgram	character [1]	''
XcomputedVarsProg...	character [1]	''
XcomputedVars	character [1]	''
names	character [13]	'BAD' 'LOAN' 'MORTDUE' 'VALUE' 'REASON' 'JOB' ...
compcomp	logical [1]	FALSE

The command below downloads the data into R List Object

```
hmeq_fromcas1 <- to.casDataFrame(hmeq_fromcas)
```

Name	Type	Value
hmeq_fromcas1	list [5960 x 13] (swat::casDat	A data.frame with 5960 rows and 13 columns
BAD	double [5960]	1 1 1 1 0 1 ...
LOAN	double [5960]	1100 1300 1500 1500 1700 1700 ...
MORTDUE	double [5960]	25860 70053 13500 NaN 97800 30548 ...
VALUE	double [5960]	39025 68400 16700 NaN 112000 40320 ...
REASON	character [5960]	'HomeImp' 'HomeImp' 'HomeImp' '' 'HomeImp' 'HomeImp' ...
JOB	character [5960]	'Other' 'Other' 'Other' '' 'Office' 'Other' ...
YOJ	double [5960]	10.5 7.0 4.0 NaN 3.0 9.0 ...
DEROG	double [5960]	0 0 0 NaN 0 0 ...
DELINQ	double [5960]	0 2 0 NaN 0 0 ...
CLAGE	double [5960]	94.4 121.8 149.5 NaN 93.3 101.5 ...
NINQ	double [5960]	1 0 1 NaN 0 1 ...
CLNO	double [5960]	9 14 10 NaN 14 8 ...
DEBTINC	double [5960]	NaN NaN NaN NaN NaN 37.1 ...

The command below downloads the data into an R Data Frame Object

```
hmeq_fromcas2 <- data.frame(to.casDataFrame(hmeq_fromcas))
```

Filter

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
1	1	1100	25860	39025.0	HomeImp	Other	10.5	0	0	94.36667	1	9	NaN
2	1	1300	70053	68400.0	HomeImp	Other	7.0	0	2	121.83333	0	14	NaN
3	1	1500	13500	16700.0	HomeImp	Other	4.0	0	0	149.46667	1	10	NaN
4	1	1500	NaN	NaN			NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	0	1700	97800	112000.0	HomeImp	Office	3.0	0	0	93.33333	0	14	NaN
6	1	1700	30548	40320.0	HomeImp	Other	9.0	0	0	101.46600	1	8	37.1136136
7	1	1800	48649	57037.0	HomeImp	Other	5.0	3	2	77.10000	1	17	NaN
8	1	1800	28502	43034.0	HomeImp	Other	11.0	0	0	88.76603	0	8	36.8848941
9	1	2000	32700	46740.0	HomeImp	Other	3.0	0	2	216.93333	1	12	NaN
10	1	2000	NaN	62250.0	HomeImp	Sales	16.0	0	0	115.80000	0	13	NaN
11	1	2000	22608	NaN			18.0	NaN	NaN	NaN	NaN	NaN	NaN
12	1	2000	20627	29800.0	HomeImp	Office	11.0	0	1	122.53333	1	9	NaN
13	1	2000	45000	55000.0	HomeImp	Other	3.0	0	0	86.06667	2	25	NaN
14	0	2000	64536	87400.0		Mgr	2.5	0	0	147.13333	0	24	NaN
15	1	2100	71000	83850.0	HomeImp	Other	8.0	0	1	123.00000	0	16	NaN
16	1	2200	24280	34687.0	HomeImp	Other	NaN	0	1	300.86667	0	8	NaN
17	1	2200	90957	102600.0	HomeImp	Mgr	7.0	2	6	122.90000	1	22	NaN

Showing 1 to 19 of 5,960 entries, 13 total columns

Using R summary() function on the downloaded data

```
summary(hmeq_fromcas2)
```

```
Selecting by Frequency
      BAD              LOAN          MORTDUE          VALUE          REASON          JOB           YOJ        
 Min.   :0.0000   Min.   : 1100   Min.   :  2063   Min.   :  8000   DebtCon:3928   Other  :2388   Min.   :  0.000  
 1st Qu.:0.0000   1st Qu.:11100   1st Qu.: 46268   1st Qu.: 66069   HomeImp:1780   ProfExe:1276   1st Qu.:  3.000  
 Median :0.0000   Median :16300   Median : 65019   Median : 89236   NA's   : 252   Office : 948   Median :  7.000  
 Mean   :0.1995   Mean   :18608   Mean   : 73761   Mean   :101776                  Mgr    : 767   Mean   :  8.922  
 3rd Qu.:0.0000   3rd Qu.:23300   3rd Qu.: 91491   3rd Qu.:119832                  Self   : 193   3rd Qu.: 13.000  
 Max.   :1.0000   Max.   :89900   Max.   :399550   Max.   :855909                  NA's   : 279   Max.   : 41.000  
                                  NA's   :   518   NA's   :   112                                 NA's   :515.000  
     DEROG             DELINQ            CLAGE             NINQ            CLNO           DEBTINC      
 Min.   :  0.0000   Min.   :  0.0000   Min.   :   0.0   Min.   : 0.000   Min.   :  0.0   Min.   :   0.5245  
 1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu.: 115.1   1st Qu.: 0.000   1st Qu.: 15.0   1st Qu.:  29.1400  
 Median :  0.0000   Median :  0.0000   Median : 173.5   Median : 1.000   Median : 20.0   Median :  34.8183  
 Mean   :  0.2546   Mean   :  0.4494   Mean   : 179.8   Mean   : 1.186   Mean   : 21.3   Mean   :  33.7799  
 3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.: 231.6   3rd Qu.: 2.000   3rd Qu.: 26.0   3rd Qu.:  39.0031  
 Max.   : 10.0000   Max.   : 15.0000   Max.   :1168.2   Max.   :17.000   Max.   : 71.0   Max.   : 203.3121  
 NA's   :708.0000   NA's   :580.0000   NA's   : 308.0   NA's   :510.000   NA's   :222.0   NA's   :1267.0000  
> 
```

Similarly in SAS Viya

```
proc summary data=mycaslib.hmeq print min q1 median mean q3 max;
   var _numeric_;
run;
proc freq data=mycaslib.hmeq;
   table _character_;
run;
```

Yields the following output like we produced in R.

The SUMMARY Procedure

Variable	Minimum	Lower Quartile	Median	Mean	Upper Quartile	Maximum
BAD	0	0	0	0.1994966	0	1.0000000
LOAN	1100.00	11100.00	16300.00	18607.97	23300.00	89900.00
MORTDUE	2063.00	46268.00	65019.00	73760.82	91491.00	399550.00
VALUE	8000.00	66069.00	89235.50	101776.05	119831.50	855909.00
YOJ	0	3.0000000	7.0000000	8.9222681	13.0000000	41.0000000
DEROG	0	0	0	0.2545697	0	10.0000000
DELINQ	0	0	0	0.4494424	0	15.0000000
CLAGE	0	115.1031968	173.4666667	179.7662752	231.5748336	1168.23
NINQ	0	0	1.0000000	1.1860550	2.0000000	17.0000000
CLNO	0	15.0000000	20.0000000	21.2960962	26.0000000	71.0000000
DEBTINC	0.5244992	29.1400314	34.8182618	33.7799153	39.0031406	203.3121487

The FREQ Procedure

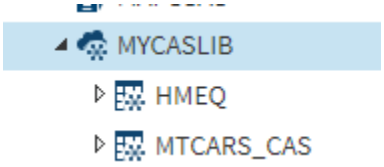
REASON	Frequency	Percent	Cumulative Frequency	Cumulative Percent
DebtCon	3928	68.82	3928	68.82
HomeImp	1780	31.18	5708	100.00
Frequency Missing = 252				

JOB	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Mgr	767	13.50	767	13.50
Office	948	16.69	1715	30.19
Other	2388	42.03	4103	72.22
ProfExe	1276	22.46	5379	94.68
Sales	109	1.92	5488	96.60
Self	193	3.40	5681	100.00
Frequency Missing = 279				

LOADING R DATA FRAMES TO SAS VIYA CAS LIBRARY

If you have data in R and would like to upload it to a SAS Viya CAS library, you can use the as.casTable() function.

```
mtcars_cas <- as.casTable(conn_binary, mtcars,
                          casOut =list(name="mtcars_cas", promote=TRUE))
```



Loading Action Sets that are not Loaded by Default

There are some action sets that are loaded by default when you load the R SWAT package but some are not. For example, the decisionTree action set is not loaded automatically but you can use the loadActionSet() function to load the desired action sets.

```
loadActionSet(conn_binary, 'decisionTree')
```

```
NOTE: Added action set 'decisionTree'.
NOTE: Information for action set 'decisionTree':
NOTE:    decisionTree
NOTE:       dtreeTrain - Trains a decision tree
NOTE:       dtreeScore - Scores a table using a decision tree model
NOTE:       dtreeSplit - Splits decision tree nodes
NOTE:       dtreePrune - Prune a decision tree
NOTE:       dtreeMerge - Merges decision tree nodes
NOTE:       dtreeCode - Generates DATA step scoring code from a decision tree model
NOTE:       forestTrain - Trains a forest
NOTE:       forestScore - Scores a table using a forest model
NOTE:       forestCode - Generates DATA step scoring code from a forest model
NOTE:       gbtreeTrain - Trains a gradient boosting tree
NOTE:       gbtreeScore - Scores a table using a gradient boosting tree model
NOTE:       gbtreeCode - Generates DATA step scoring code from a gradient boosting tree model
NOTE:       dtreeExportModel - Export the aStore model for a tree model table
```

CONCLUSIONS

You should now have a basic understanding of how the R SWAT package works and build off of the examples shown in this paper.

Using the R SWAT package with SAS Viya gives a user the flexibility of using the R programming language on data sitting in a SAS Viya CAS library. This will allow SAS and R developers to collaborate more easily using the same data sources.

CONTACT INFORMATION

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

Brian Varney
Experis Solutions
269-365-1755
brian.varney@experis.com