Getting Started with CASL Programming 3.4

Connect and Start a Session

1. If you have not already done so, download the data sets that are used in examples. You can download the data at http://support.sas.com/documentation/onlinedoc/viya/examples.htm. Put them in a directory that is accessible to SAS.
2. Open SAS Studio from the URL in the form of http://hostname:port. Sign in using your user ID and password for your operating system account.
   a. Use the **Server Files and Folders** section to navigate to the directory that has data sets.
   b. Right-click on the directory and select **Create** ⇨ **Library**. Specify **heart** as the name.
3. Use the **Libraries** section to confirm that the new library is listed.
4. Check that you have a CAS session.
   a. Enter the following line in the code editor and click to run the code.

      ```
      cas casauto list;
      ```

      If you get the error: `"ERROR: Request failed. Session CASAUTO not recognized"`, see Tip: Automatically Connect and Generate Librefs.

   b. Click the **Code** tab. The following note indicates that you have an active CAS session.

```
NOTE: The session CASAUTO is ACTIVE using port 5570 and host cloud.example.com for user sasdemo.
        The session UUID is 0ed1cc35-c3ec-df49-be6e-68b9193eb8b8.
```

About Your Connection and Server

After you connect, a session is started for you. As a documentation convention, a variable that is named Casauto is used to represent the session.

Casauto
: The name of the variable that is in CASL. It represents the session that is started for you in SAS Cloud Analytic Services.

session
: The software process that is started on the same hosts as SAS Cloud Analytic Services. When you reference your session through the Casauto variable, statistical computations and actions are run on the server.

As soon as you connect, a good practice is to print information about the connection and session:

```
proc cas;
   session casauto;
   session.sessionStatus result=r;
   print "Session status:";
   print "State:         " r["state"];
   print "Connections:   " r["number of Connections"];
   print "Timeout:       " r["Timeout"] "minutes";
   print "Action Status: " r["ActionStatus"];
   print "Authenticated: " r["Authenticated"];
   print "Locale:        " r["locale"];
run;
```

The following is printed to the SAS log.

Output 1 *SAS Log*

```
Session status:
 State:         Connected
 Connections:   1
 Timeout:       60 minutes
 Action Status: Action is active
 Authenticated: Yes
 Locale:        en_US
```

Your results show different values. In the event of a network interruption between CAS and the server, the UUID for the session can be used to reconnect to a session.

CASL is designed to create arguments that are used to submit actions to the CAS server. You can use the serverStatus action to retrieve status information of the server.

```
proc cas;
   session casauto;          /*1*/
   builtins.serverStatus;    /*2*/
run;
```

1 The SESSION statement indicates which CAS session to use for running statements that follow.

2 The serverStatus action provides details about the server and the status of all nodes.

Output 2 *Results: serverStatus*

Results from builtins.serverStatus

Node Count	Total Actions
143	5

Node Name	Role	Uptime (Sec)	Running	Stalled
Computer 1	backup ctl	1523.807	0	0
Computer 2	worker	1523.807	0	0
Computer 3	worker	1523.807	0	0
Computer 4	worker	1523.807	0	0
Computer 5	worker	1523.808	0	0

Example: Working with CAS Actions and Results

About the Example

The following example uses the CSV file, Hmeq, which contains data about applicants who are granted credit for a home equity loan. Download the CSV file: http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/hmeq.csv. The example uses CAS actions and CASL statements to explore and analyze the data.

Start a CAS Session

Use the CAS statement to start a CAS session with default properties. You can set the system options, CASHOST= and CASPORT= to a host and port that are valid for your site.

```
options cashost="cloud.example.com" casport=5570;
```

If necessary, create session Casauto.

```
cas casauto;
```

You can also view your active sessions.

Load the Data

```
proc cas;                                                    /* 1 */
   upload path='C:\Users\sasdemo\hmeq.csv'                   /* 2 */
      casOut={
      name="hmeq"
      replace=True
   }
   importOptions={fileType="csv"}
;
run;
```

1 The PROC CAS statement enables you to submit actions from the SAS client to the CAS server.

2 The UPLOAD statement transfers the Hmeq.csv file from the directory that SAS can access to the server. After the transfer, the server loads the data from the Hmeq.csv file into an in-memory table.

The following is printed to the SAS log.

Output 3 *SAS Log*

```
NOTE: Active Session now CASAUTO.
NOTE: Cloud Analytic Services made the uploaded file available as table HMEQ in caslib CASUSER.
NOTE: The table HMEQ has been created in caslib CASUSER from binary data uploaded to Cloud
      Analytic Services.
{caslib=CASUSER, tableName=HMEQ}
```

Explore the Data Using CAS Actions

```
proc cas;
   table.columnInfo / table='hmeq';                                    /* 1 */
   simple.summary result=rSum / table={name='hmeq', groupby='bad'};    /* 2 */
      subSet={"MAX","MIN","MEAN","N"};
   describe rSum;                                                      /* 3 */
   print rSum["ByGroup2.Summary". 3:5];                                /* 4 */
run;
```

1 The columnInfo action lists column information for the Hmeq table. Examples are name, length, and data type.

2 The Summary action generates the descriptive statistics of all the interval variables in the data set. The descriptive statistics are grouped by the binary variable Bad. Bad identifies a client who has either defaulted or repaid their home equity loan. The results of the Summary action are saved in the rSum variable.

3 The DESCRIBE statement lists the structure and data type of rSum.

4 The PRINT statement writes the values of the variable rSum to the output destination.

Output 4 *Results: columnInfo Action*

Results from table.columnInfo

Column Information for HMEQ in Caslib MYLIB						
Column	Id	Type	Length	Formatted Length	Format Width	Format Decimal
BAD	1	double	8	12	0	0
LOAN	2	double	8	12	0	0
MORTDUE	3	double	8	12	0	0
VALUE	4	double	8	12	0	0
REASON	5	varchar	7	7	0	0
JOB	6	varchar	7	7	0	0
YOJ	7	double	8	12	0	0
DEROG	8	double	8	12	0	0
DELINQ	9	double	8	12	0	0
CLAGE	10	double	8	12	0	0
NINQ	11	double	8	12	0	0
CLNO	12	double	8	12	0	0
DEBTINC	13	double	8	12	0	0

Output 5 *SAS Log: rSum Variable*

```
dictionary ( 3 entries, 3 used);
 [ByGroupInfo] Table ( [2] Rows [3] columns
Column Names:
 [1] BAD               [                ] (double)
 [2] BAD_f             [                ] (char)
 [3] _key_             [                ] (varchar)
 [ByGroup1.Summary] Table[Summary] ( [11] Rows [5] columns
Column Names:
 [1] Column            [Analysis Variable] (char)
 [2] Min               [Minimum         ] (double) [D8.4]
 [3] Max               [Maximum         ] (double) [D8.4]
 [4] N                 [N               ] (double) [BEST10.]
 [5] Mean              [Mean            ] (double) [D8.4]
 [ByGroup2.Summary] Table[Summary] ( [11] Rows [5] columns
Column Names:
 [1] Column            [Analysis Variable] (char)
 [2] Min               [Minimum         ] (double) [D8.4]
 [3] Max               [Maximum         ] (double) [D8.4]
 [4] N                 [N               ] (double) [BEST10.]
 [5] Mean              [Mean            ] (double) [D8.4]
```

Output 6 *Results: Summary Action of GroupBy Variable: Bad*

ByGroup2.Summary: Results

Descriptive Statistics for HMEQ				
Column	Minimum	Maximum	N	Mean
MORTDUE	2063.00	399550	1083	69460
VALUE	8800.00	855909	1084	98173
YOJ	0	41.0000	1124	8.0278

Analyze and Save Results

```
proc cas;
   simple.summary result=xSumm / table={name='hmeq'};    /*1*/
      subSet={"MAX","MIN","MEAN","N"};
   val=findtable(xSumm);                                 /*2*/
   saveresult val dataout=work.summary;                  /*3*/
run;
```

1 The Summary action generates descriptive statistics of numeric variables of the Hmeq table and saves the results in the xSumm variable. Examples are sample mean, sample variance, and sample size.

2 The FINDTABLE function returns the first result table in the xSumm variable and the result table is saved in the Val variable.

3 The SAVERESULT statement saves the result table Val as a data set named Work.Summary.

The following is printed to the SAS log.

Output 7 *SAS Log*

```
NOTE: The data set work.summary has 11 observations and 17 variables.
```

See Also

- “CAS Statement” in *SAS Cloud Analytic Services: User’s Guide*
- “CASHOST= System Option” in *SAS Cloud Analytic Services: User’s Guide*
- “CASPORT= System Option” in *SAS Cloud Analytic Services: User’s Guide*
- “Column information” in *SAS Viya: System Programming Guide*
- “DESCRIBE Statement” in *SAS Cloud Analytic Services: CASL Reference*
- “FINDTABLE Function” in *SAS Cloud Analytic Services: CASL Reference*
- “PRINT Statement” in *SAS Cloud Analytic Services: CASL Reference*
- “SAVERESULT Statement” in *SAS Cloud Analytic Services: CASL Reference*
- “Summary” in *SAS Visual Analytics: Programming Guide*
- “UPLOAD Statement” in *SAS Cloud Analytic Services: CASL Reference*

Advanced Example: Train Gradient Boosted Trees with k-fold Cross Validation

About the Example

The purpose of this example is to describe how to train gradient boosted trees with k-fold cross validation. A k-fold cross validation process finds the average estimated validation error (misclassification error for nominal targets or average square error for interval targets) for the trained model. During cross validation, all data are divided into k subsets (folds). For each fold, a new model is trained, and then validated using the selected fold. The validation error estimates are then averaged over each set of training and scoring executions to obtain a single value.

Load the Data

Create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server.

```
libname mycas mysess;      /* 1 */
data mycas.heart;          /* 2 */
   set sashelp.heart;
run;
```

1 Define your own session and CAS engine librefs that connect to the CAS server. The CAS statement creates the CAS session named *mysess*, and the LIBNAME statement creates the *mycas* CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

2 The DATA step creates the data table *mycas.heart*, which consists of 5209 observations that have 17 variables. This DATA step assumes that your CAS engine libref is named *mycas*, but you can substitute any appropriately defined CAS engine libref.

Explore Data Using CAS Actions

This section describes how to access and manage your data using the tables action set. For more information on the table action set see *SAS Viya: System Programming Guide*.

```
proc cas ;
   tableinfo /table='heart';        /* 1 */
run;

columninfo result=r /table='heart'; /* 2 */
print r;
run;

simple.summary/table='heart';       /* 3 */
run;
```

1 The table action set executes the tableInfo action, which shows information about a table.

2 The table action set executes the columnInfo action, which shows column information.

3 The simple action set executes the summary action, which generates descriptive statistics of numeric variables such as the sample mean, sample variance, sample size, sum of squares, and so on. For more information about the simple action set, see "Simple Analytics Action Set" in *SAS Visual Analytics: Programming Guide*

Output 8 *Results: Table Information for Heart*

Table Information for Caslib CASUSERHDFS									
Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Duplicated Rows	View	Compressed
HEART	5209	17	utf-8	09Sep2016:15:18:00	09Sep2016:15:18:00	No	No	No	No

Output 9 *Results: Column Information for Heart*

Results from table.tableInfo

Table Information for Caslib CASUSERHDFS									
Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Duplicated Rows	View	Compressed
HEART	5209	17	utf-8	09Sep2016:15:18:00	09Sep2016:15:18:00	No	No	No	No

Output 10 *Results: Simple Summary for Heart*

Descriptive Statistics for HEART														
Column	Min	Max	N	Sum	Mean	Std Dev.	Std Error	Variance	Coefficient of Variation	Corrected Sum of Squares	Uncorrected Sum of Squares	t Value	Pr > |t|	Number Missing
AgeCHDdiag	32.0000	90.0000	1449	91726	63.3030	10.0182	0.2632	100.36	15.8258	145328	5951856	240.53	<.0001	3760
AgeAtStart	28.0000	62.0000	5209	229554	44.0687	8.5750	0.1188	73.5298	19.4581	382943	10499096	370.92	<.0001	0
Height	51.5000	76.5000	5203	337223	64.8132	3.5827	0.04967	12.8358	5.5277	66772	21923268	1304.91	<.0001	6
Weight	67.0000	300.00	5203	796510	153.09	28.9154	0.4009	836.10	18.8883	4349402	1.2628E8	381.89	<.0001	6
Diastolic	50.0000	160.00	5209	444633	85.3586	12.9731	0.1797	168.30	15.1983	876512	38829767	474.88	<.0001	0
Systolic	82.0000	300.00	5209	713162	136.91	23.7396	0.3289	563.57	17.3396	2935064	1.0057E8	416.23	<.0001	0
MRW	67.0000	268.00	5203	624139	119.96	19.9834	0.2770	399.34	16.6587	2077348	76947517	433.00	<.0001	6
Smoking	0	60.0000	5173	48453	9.3665	12.0315	0.1673	144.76	128.45	748677	1202513	55.99	<.0001	36
AgeAtDeath	36.0000	93.0000	1991	140438	70.5364	10.5594	0.2366	111.50	14.9701	221887	10127880	298.06	<.0001	3218
Cholesterol	96.0000	568.00	5057	1150050	227.42	44.9355	0.6319	2019.20	19.7590	10209082	2.7175E8	359.90	<.0001	152

Generate Folds for Cross Validation

Use CASL to partition the table and specify the number of partition folds for cross validation. For this example, we are generating ten folds. A column named *_fold_* is produced, and the only way to get repeatable folds is to copy the new table and add the *_fold_* column.

```
partition / table={name='heart',
   compvars={'_fold_'},
   comppgm='call streaminit(__rankid*1000);_fold_=floor(rand("UNIFORM")*10);'}
      outtable={name='new_heart_with_fold', replace=True
};
run;
```

Output 11 *Log Output: Partitioned Table Log Output*

```
{caslib=CASUSERHDFS(casuser),tableName=NEW_HEART_WITH_FOLD,rowsTransferred=27,
shuffleWaitTime=0.0000758171,minShuffleWaitTime=0,maxShuffleWaitTime=1.9073486E-6,
averageShuffleWaitTime=4.5672957E-7}
```

Verify the _Fold_ Column

In order to verify the new *_fold_* column, perform these simple statistics.

```
summary/ table='new_heart_with_fold' inputs={'_fold'};
run;
simple.distinct / table={name='new_heart_with_fold', inputs='_fold_'};
run;
freq/ table={name='new_heart_with_fold', inputs={'_fold_'}};
run;
columninfo result=r /table={name='new_heart_with_fold'};
run;
```

Output 12 *Results: Simple Distinct Results*

Distinct Counts for NEW_HEART_WITH_FOLD

Column	Number of Distinct Values	Number of Missing Values	Truncated
fold	10	0	No

Output 13 *Results: Frequency Table*

Frequency for NEW_HEART_WITH_FOLD				
Column	Numeric Value	Formatted Value	Level	Frequency
fold	0	0	1	508
fold	1	1	2	524
fold	2	2	3	538
fold	3	3	4	520
fold	4	4	5	513
fold	5	5	6	530
fold	6	6	7	550
fold	7	7	8	525
fold	8	8	9	490
fold	9	9	10	511

Remove the _Fold_ Column

Remove the _fold_ column since it is not our analysis variable.

```
nVars=dim(r['columninfo'])-1;
```

Create an Input Variable List

Create a variable list where you define each variable. Each variable is defined by an expression, character, or numeric value. For this example, we use CASL to create our variable list. The syntax for the variable list follows the Assignment statement syntax. For more information see "Assignment Statement" in *SAS Cloud Analytic Services: CASL Reference*.

```
i=4;
j=2;
xx= {r["columninfo"][1,1]};    /*1*/
do while (i<=nVars);           /*2*/
   xx=xx + r["columninfo"][i,1];
   i=i+1;
   j=j+1;
end;
print xx;
run;
```

1 The target *xx* uses the result variable from the previously run columninfo, and *r* as its expression in addition to searching for the value in 1, 1.

2 The DO WHILE statement executes statements in a DO loop repetitively as long as the condition is true.For more information see "Using a DO WHILE Statement " in *SAS Cloud Analytic Services: CASL Reference*.

Output 14 *Log Output: Variable List Concatenated*

```
{Status,Sex,AgeAtStart,Height,Weight,Diastolic,Systolic,MRW,Smoking,AgeAtDeath,Cholesterol,
Chol_Status,BP_Status,Weight_Status,Smoking_Status}
```

Train a Decision Tree Using a Left-Out Fold

Train the decision tree by splitting the sub-sampled data, then splitting each resulting segment, and so on recursively until some constraint is met.

```
function OneFoldTree(nFold, iFold);                          /* 1 */
/* generate _fold_ where */
   foldwhere1 = "_fold_ NE " || (String)iFold;
   foldwhere2 = "_fold_ EQ " || (String)iFold;
   mymodel = "gbt_" || (String)iFold;                        /* 2 */
   decisiontree.gbtreetrain  /                               /* 3 */
      table={name="new_heart_with_fold", where=foldwhere1}
      inputs=xx
      target="AgeAtDeath"
      casout={name=mymodel, replace=1}
      maxbranch=2
      maxlevel=8
      leafsize=60
      ntree=100
      binorder=1
      nbins=100
      seed=1234
      learningRate=0.1
      subsamplerate=0.7
      m=64
      ;
   decisionTree.gbtreescore result = r/ table={name='new_heart_with_fold',
      where=foldwhere2} model={name=mymodel};                /* 4 */
   print r;
   myscoredata = "gbtscore_" || (String)iFold;               /* 5 */
      saveresult r  replace dataset=myscoredata;             /* 6 */
/* return prediction error; MSE or missclassification rate */
   return (r["ScoreInfo"][3,2]);                             /* 7 */
end func;
```

1 The FUNCTION statement creates a new function that can be called in an expression. In this example, the function is named *OneFoldTree* and has two arguments named *nFold* and *iFold*. For more information about the FUNCTION statement syntax, see "FUNCTION Statement" in *SAS Cloud Analytic Services: CASL Reference*.

2 Generate a model name.

3 Create a model without iFold, and score on the holdout iFold using the train model. The action set decision tree executes the action gbtreetrain that trains a gradient boosting tree. This function can be easily expanded to train other models such as a neural network with cross validation. For more information about gbtreetrain action syntax, see "Trains gradient boosting tree" in *SAS Visual Analytics: Programming Guide*.

4 Score iFold-th data with a trained model. The action set decision tree executes the action gbtreescore that scores a table using a gradient boosting tree model. For more information about gbtreescore action syntax, see "Scores a table using gradient boosting tree" in *SAS Visual Analytics: Programming Guide*.

5 Generate a score data set name. In this example the score data set name is *myscoredata* . The data set name is evaluated as `"gbtscore_" || (String)iFold`. For more information about the Assignment statement syntax, see "Assignment Statement" in *SAS Cloud Analytic Services: CASL Reference*.

6 The SAVERESULT statement creates a SAS data set from the results of an ACTION. For more information about the SAVERESULT statement syntax, see "SAVERESULT Statement" in *SAS Cloud Analytic Services: CASL Reference*.

7 The RETURN statement returns a value from the current function. For more information about the RETURN statement syntax, see "RETURN Statement" in *SAS Cloud Analytic Services: CASL Reference*.

Output 15 *Partial Results: Decision Tree Action*

The SAS System

Results from decisionTree.gbtreeTrain

Gradient Boosting Tree for NEW_HEART_WITH_FOLD	
Number of Trees	100.000000
Distribution	1.000000
Learning Rate	0.100000
Subsampling Rate	0.700000
Number of Selected Variables (M)	14.000000
Number of Bins	100.000000
Number of Variables	14.000000
Max Number of Tree Nodes	33.000000
Min Number of Tree Nodes	17.000000
Max Number of Branches	2.000000
Min Number of Branches	2.000000
Max Number of Levels	8.000000
Min Number of Levels	6.000000
Max Number of Leaves	17.000000
Min Number of Leaves	9.000000
Maximum Size of Leaves	466.000000
Minimum Size of Leaves	60.000000
Random Number Seed	1234.000000

Output CAS Tables			
CAS Library	Name	Number of Rows	Number of Columns
CASUSERHDFS	gbt_0	2736	29

Run Each Model In Its Own Session

You can run each model in its own session by creating a function for each model.

```
function KFoldCV(nFold);                          /* 1 */
   do i=1 to nFold;                               /* 2 */
      myerror[i] = OneFoldTree(nFold, i);
   end;
   return (myerror);                              /* 3 */
end func;
```

1 The FUNCTION statement creates a new function that can be called in an expression. In this example, the function is named *KFoldCV* and has one argument named *nFold*.

2 The DO statement, Iterative iterates over the list that starts at the value of *1* to the value of *nFold*. For more information about the DO Iterative syntax, see "DO Statement, Iterative" in *SAS Cloud Analytic Services: CASL Reference*.

3 The RETURN statement returns a value from the current function. For more information about the RETURN statement syntax, see "RETURN Statement" in *SAS Cloud Analytic Services: CASL Reference*.

Train k (*n*Fold) Models

During cross validation, all data are divided into k subsets (folds). For each fold, a new model is trained then validated using a selected fold. In this example, we have ten folds that we are going to train against the selected fold (*n*Fold=1).

```
nFold = 10;
ModelError = KFoldCV(nFold);
run;
```

Output 16 *Log Output: Trained Models*

```
NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
58.744875114

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
50.51471611

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
71.307144715

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
59.846180435

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
63.013744439

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
59.198893492

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
60.686242454

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
59.041830014

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
62.895729306

NOTE:  The data set work.myscoredata  has 3 observations and 2 variables.
NOTE:  The data set work.myscoredata1  has 100 observations and 6 variables.
57.160065856
```

```
/*print error into log*/
print ModelError;
run;
```

Output 17 *Output Log: Model Error for Each Fold*

```
{58.744875114,50.51471611,71.307144715,59.846180435,63.013744439,59.198893492,60.686242454,
59.041830014,62.895729306,57.160065856}
```

Compute Average Error Rate from Cross Validation

This section describes how to compute the average error rate or the misclassification error or average square of k-fold cross validation.

```
mse  = 0;           /*1*/
do i= 1 to nFold;   /*2*/
   mse = mse + ModelError[i];
end;
run;
```

1 Compute the average error rate from cross validation. The mean squared error (MSE) of an estimator measures the average of the squares of the deviations.

2 The DO statement, Iterative iterates over the list that starts at the value of *1* to the value of *nFold*. For more information about the DO statement, Iterative syntax see "DO Statement, Iterative" in *SAS Cloud Analytic Services: CASL Reference*.

Output 18 *Log Output: Conversion of String to Number*

```
NOTE: String '58.744875114' convert to number.
NOTE: String '50.51471611' convert to number.
NOTE: String '71.307144715' convert to number.
NOTE: String '59.846180435' convert to number.
NOTE: String '63.013744439' convert to number.
NOTE: String '59.198893492' convert to number.
NOTE: String '60.686242454' convert to number.
NOTE: String '59.041830014' convert to number.
NOTE: String '62.895729306' convert to number.
NOTE: String '57.160065856' convert to number.
```

```
mse = mse/nFold;    /*1*/
rmse = sqrt(mse);
print "mse=" mse "rmse=" rmse;
run;
```

1 Print the average mean squared error (MSE) and root mean squared error (RMSE) to the log.

Output 19 *Log Output: Average MSE and RMSE*

```
mse=60.240942194 rmse=7.7615038616
```