Paper 3473-2019

Understanding, Tracking, and Visualizing Decisions with SAS® Viya®

Carl Sommer, SAS Institute Inc.

ABSTRACT

SAS® Decision Manager on SAS® Viya® provides more than just decision results; it also provides insight into how a decision was made. Macros are available for use with SAS Decision Manager 5.2, which enable the SAS® Studio user to work with information about which decision paths were taken as well as what rules were fired, thus providing organizations with analytical insights to revise their business' decision-making processes with a precision not available with many other tools. This paper shows how to use these macros along with SAS® Cloud Analytic Services (CAS) to understand, track, and visualize the behavior of their organization's decisions.

INTRODUCTION

SAS Decision Manager empowers users to author, test, and publish decisions for their organization. These decisions consist of models, rulesets, custom code, and condition nodes to control the flow of logic provided by the user. The visual user interface and REST APIs provide a method for the user to manage the life cycle of a decision.

In the 2016 SAS Global Forum paper "Taming the Rule," Crain and Upton establish the case for performing analyses based on rule-fired information, and provide examples using SAS® Business Rules Manager and SAS® Visual Analytics. This paper builds on that foundation, using SAS Decision Manager, SAS Studio, and the SAS Viya architecture.

Using a set of macros (available from the conference code samples area), SAS Studio users have the ability to execute decisions and analyze the results. Unlike the Scoring functionality in the visual user interface, which works only with data in SAS Cloud Analytic Services (CAS) tables, these macros can leverage data from SAS tables, from tables that are accessed via SAS/ACCESS engines, and from CAS tables.

This paper will highlight the following uses of these macros:

1. Execution of a decision or ruleset in SAS Studio.
2. Extracting rule-fired information from the results of the execution of a decision or ruleset.
3. Extracting path-tracking and node-activity information from the execution of a decision.
4. Analysis and exploration of the decision.

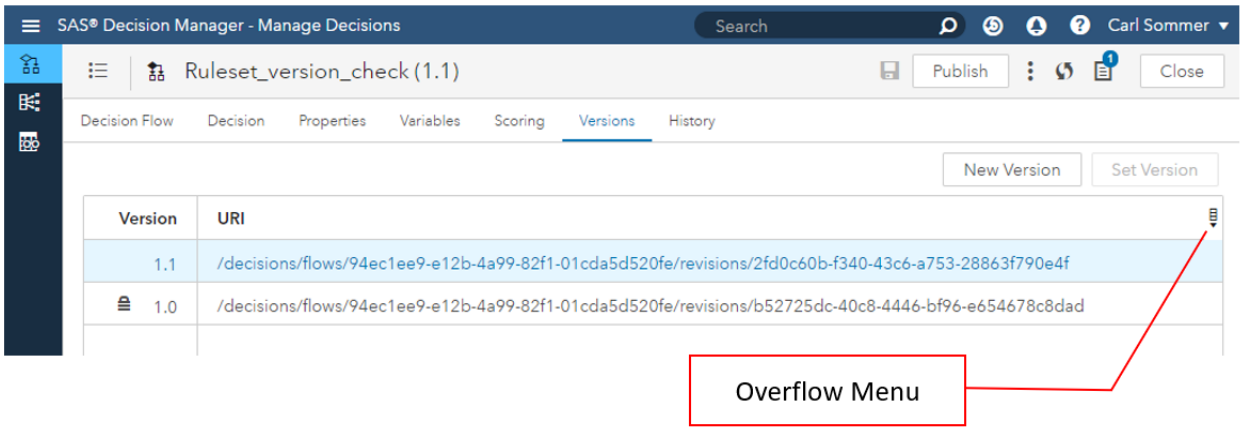
In addition, we present special considerations for using these macros with CAS, and producing reports with SAS Visual Analytics.

SPECIFYING THE RULESET OR DECISION

SAS Decision Manager allows the user to maintain multiple versions of their content. It is therefore essential that the user associate the decision or ruleset of interest with the macros that will be deployed, including the specific revision of the decision or ruleset. A URI (universal resource indicator) is used for this purpose.

URI information is available on the **Versions** tab of a decision; however, it is not shown by default. Make the URI column visible by selecting **Manage Columns** from the Overflow

menu and selecting the **URI** column to display. Display 1 shows the location of the overflow menu, and the resulting view has the **Version** and **URI** columns displayed.



Display 1. Version and URI Displayed in Versions Tab

As an alternative, the user could obtain the URI by using the SAS Decision Manager REST APIs, which are documented at https://developer.sas.com/apis/rest/DecisionManagement/.

The URI is a required parameter for all the macros discussed in this paper.

EXECUTING A RULESET OR DECISION

Once the URI of the specific version of the ruleset or decision is known, the appropriate execution macro can be used to process an input table to produce an output table of results.

The macro `%dcm_execute_ruleset` is used to execute a ruleset, and the `%dcm_execute_decision` macro is used to execute a decision. The output table from these execution macros is then used as input to subsequent macros for rule-fired and path-tracking analysis.

Note that the execution macros do not support mapping of input columns to ruleset or decision terms. This means that any table that you use as input must have column names and data types that match the parameters that the ruleset or decision expects for input.

The code in Example 1 uses data from the MYDATA.TRANSACTIONS table to execute the indicated decision and produce the output table WORK.DECISION_RESULTS:

```
/* apologies for the line-wrapping of the URI to fit the page */
%let URI=%str(/decisions/flows/4f073334-01f4-4586-bee6-
5f4630c5ff7f/revisions/46529b1d-939a-43bc-b226-d53b9855d2f4);

%dcm_execute_decision(
    URI=&URI,
    inputTable=MYDATA.TRANSACTIONS,
    outputTable=WORK.DECISION_RESULTS)
```

Example 1. Executing a Decision

EXTRACTING RULE-FIRED INFORMATION

Each output row from the execution of a decision or ruleset includes information about the rules that were fired to produce that row. This information is encoded in a column called **RULEFIREDFLAGS**. The decoding of this information into individual rule-fired records is specific to the URI for the decision or ruleset. A single execution output record could be a result of zero or multiple rules being fired.

The code in Example 2 uses data from the table WORK.DECISION_RESULTS to produce the table WORK.RULEFIRE_DETAIL:

```
%dcm_rulefire_detail(URI=&URI, /* URI from prior EXECUTE example*/
    inputTable=WORK.DECISION_RESULTS, outputTable=WORK.RULEFIRE_DETAIL)
```

Example 2. Generating Rule-Fired Detail Data

All three of the parameters shown (**URI=, inputTable=,** and **outputTable=**) are required.

Table 1 shows the columns in the resulting rule-fired detail table:

Column	Notes
_recordCorrelationKey	This can be used to join the potentially multiple rule-fired detail records to the originating execution output record.
RULE_NM	Name of the rule that fired
RULE_ID	UUID of the rule that fired
RULE_SET_NM	Name of the rule set that the rule belongs to
RULE_SET_ID	UUID of the rule set that the rule belongs to
RULE_SET_VERSION_ID	UUID of the version of the rule set
RULE_ACTION_FIRE_ID	Unique key for this rule-fired event
ruleFiredSequence	Numeric sequence in which this rule was fired for this execution output record
The following additional fields are created if URI= specified a decision instead of a ruleset:	
DECISION_NAME	Name of the decision
DECISION_ID	UUID of the decision
DECISION_VERSION_ID	UUID of the version of the decision
NODE_ID	UUID of the node in the decision

Table 1. Rule-Fired Detail Table Columns

RULE-FIRED SUMMARY

The %dcm_rulefire_summary macro summarizes the rule-fired detail table by RULE_ID. It produces a total **ruleFiredCount** column for each specific rule, along with the identifying name and ID columns shown in Table 1, minus the detail fields of _recordCorrelationKey, RULE_ACTION_FIRE_ID, and ruleFiredSequence.

The code in Example 3 uses data from the table WORK.RULEFIRE_DETAIL to produce the table WORK.RULEFIRE_SUMMARY:

```
%dcm_rulefire_summary(URI=&URI, /* URI from prior EXECUTE example*/
    inputTable=WORK.RULEFIRE_DETAIL, outputTable=WORK.RULEFIRE_SUMMARY)
```

Example 3. Generating Rule-Fired Summary Data

DECISION PATH TRACKING

Decisions consist of a variety of node types, with the various paths through the decision directed by **Condition** nodes. Just as the output of the decision is important, so is the path that created the output. The output table that is created from executing a decision contains a column called **PathID**, which records the node ID of every node in the decision that was traversed in order to produce the output record. Three macros are provided to assist with analysis of this data by formatting it in multiple ways for purposes of understanding:

- Paths traversal
- Decision structure
- Individual node visit counts

PATH TRAVERSAL

At the heart of the question "How did this decision get made?" is the question "What path was taken to reach this result?" Although some decisions take nearly straight-line paths, complex decisions can have numerous unique paths from start to finish.

The `%dcm_decision_path_frequency` macro reads the output table from the execution of a decision to produce a table. The resulting table contains the unique set of traversed paths to produce the output, and the number of times that the path was taken.

The code in Example 4 uses data from the table WORK.DECISION_RESULTS to produce the table WORK.PATH_FREQ:

```
%dcm_decision_path_frequency(URI=&URI, /* URI from prior EXECUTE example*/
    inputTable=WORK.DECISION_RESULTS, outputTable=WORK.PATH_FREQ)
```

Example 4. Generating Path Traversal Frequency Data

Note that the output table from this macro does *not* include the set of paths that were never traversed.

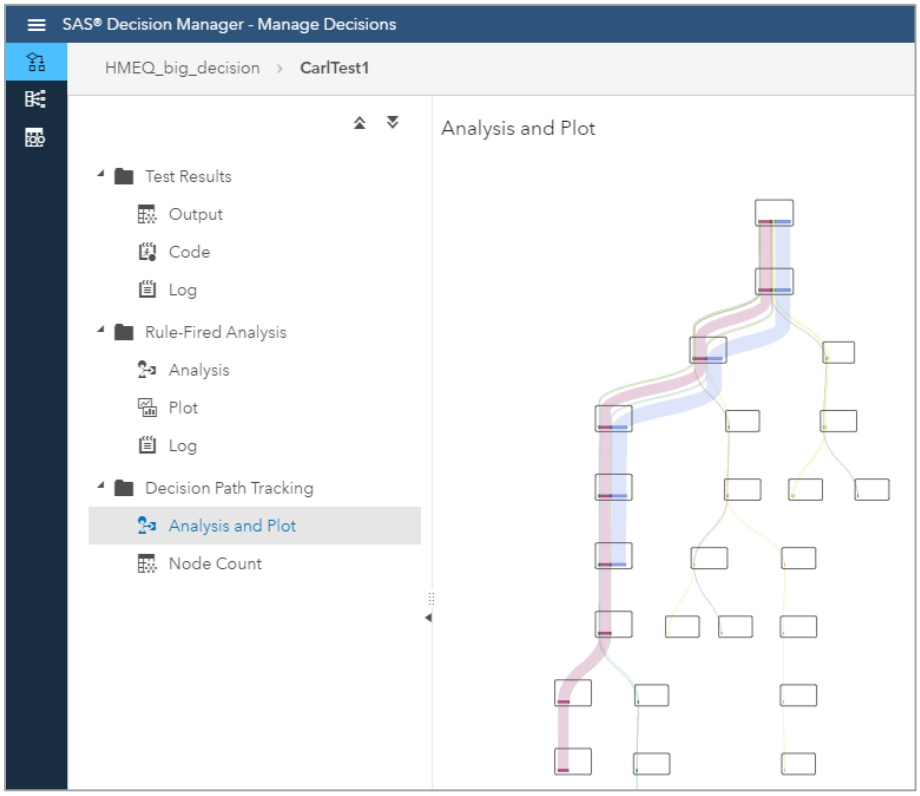
Display 2 shows the path frequency output table. This table includes a **PathID** column, which lists the node ID for each node that was traversed, separated by a slash (/), and the **Count** column, which indicates the number of times this path was traversed.

PathID	Count
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef/4a989260-9cac-4e66-903a-f44f53a04721/24dcb390-ae7d-4f85-8508-a97bdb486ee7/95188ae9-74a9-404c-8904-12820d0f71f0/ed3fc808-d148-409a-82cd-053edd9e181e/8ed63501-73ed-4b67-aae0-40cb0b4ef986/94e3112f-0876-466b-ad1f-b5b56097bb14/efbb7297-c27f-4f57-86b9-db9456356bb5/4e2fdfb4-da7a-4082-b96b-9b4eb167bab6	85
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef	62
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef/4a989260-9cac-4e66-903a-f44f53a04721/24dcb390-ae7d-4f85-8508-a97bdb486ee7/95188ae9-74a9-404c-8904-12820d0f71f0/ed3fc808-d148-409a-82cd-053edd9e181e/8ed63501-73ed-4b67-aae0-40cb0b4ef986/94e3112f-0876-466b-ad1f-b5b56097bb14/efbb7297-c27f-4f57-86b9-db9456356bb5/4777308d-6d90-4496-a4a4-5011c195dfe5	18
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/1820f3bd-1bb9-4308-9ef0-10da2100c5c2/75fbe352-0869-4aab-9d75-d8558e7405ea/d6d58851-9bb5-4171-b946-33bedc73f13e/b45ce285-d625-491b-8c15-0d4d957b4a91	4
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/1820f3bd-1bb9-4308-9ef0-10da2100c5c2/75fbe352-0869-4aab-9d75-d8558e7405ea/d6d58851-9bb5-4171-b946-33bedc73f13e/9e01d5c2-3fa0-4dc2-8d8d-beebce207bdc	37
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef/4a989260-9cac-4e66-903a-f44f53a04721/24dcb390-ae7d-4f85-8508-a97bdb486ee7/95188ae9-74a9-404c-8904-12820d0f71f0/16037219-ebf2-4669-a22a-c65e8278b138/51316c93-cd44-4c17-990b-02c96da96112	1197
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/a489974c-86e6-403e-a073-e3519add234a/ad9378d5-3605-4fb5-aec8-e99362482dfb/1755a970-1e47-4bcc-a743-792b20366795	15
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef/4a989260-9cac-4e66-903a-f44f53a04721/24dcb390-ae7d-4f85-8508-a97bdb486ee7/95188ae9-74a9-404c-8904-12820d0f71f0/16037219-ebf2-4669-a22a-c65e8278b138	99

Display 2. Path Frequency Table

DECISION STRUCTURE

The traversed path is decomposed into a table that stores the depth of the node in the decision structure and the sequence of the node in the traversal. This information can be used to reconstruct the dependency graph nature of the decision. This table is used internally to produce the Sankey diagram that is available in the SAS Decision Manager UI, shown in Display 3.



Display 3. Sankey Path Tracking Plot in User Interface

The `%dcm_decision_path_nodes` macro reads the output table from the `%dcm_decision_path_frequency` macro to produce a table that records the structure of the decision.

The code in Example 5 uses data from the table WORK.PATH_FREQ to produce the table WORK.PATH_NODES:

```
%dcm_decision_path_nodes(URI=&URI, /* URI from prior EXECUTE example*/
    inputTable=WORK.PATH_FREQ, outputTable=WORK.PATH_NODES)
```

Example 5. Generating Decision Node Structure Data

This table breaks each traversal path into legs of the traversal showing the depth and sequence of traversal. Display 4 shows the value of **PathID** for the full traversal that created an output record.

PathID
/a0b3c65d-7511-441d-8573-81eff13f730d/2376568b-22ec-438b-b260-3ac66df328a2/acd70b3c-61f4-48fc-a33b-22d5a17987d6/14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef/4a989260-9cac-4e66-903a-f44f53a04721/24dcb390-ae7d-4f85-8508-a97bdb486ee7/95188ae9-74a9-404c-8904-12820d0f71f0/ed3fc808-d148-409a-82cd-053edd9e181e/8ed63501-73ed-4b67-aae0-40cb0b4ef986/94e3112f-0876-466b-ad1f-b5b56097bb14/efbb7297-c27f-4f57-86b9-db9456356bb5/4e2fdfb4-da7a-4082-b96b-9b4eb167bab6

Display 4. Full Path Traversal for an Output Record

Each of the nodes that was traversed is broken out as a separate row, as shown in Display 5. Note that this table includes additional columns for the name and ID for each of the nodes. Also recognize that the sequence values might have gaps, but the values are monotonically increasing, thus presenting the sequence.

node_id	node_group_id	node_seq_no	node_depth_no	node_type
a0b3c65d-7511-441d-8573-81eff13f730d	1bc92637-5e58-450d-882f-6fa3269e5fff	0	0	application/vnd.sas.decision.step.model
2376568b-22ec-438b-b260-3ac66df328a2	1bc92637-5e58-450d-882f-6fa3269e5fff	1	0	application/vnd.sas.decision.step.condition
acd70b3c-61f4-48fc-a33b-22d5a17987d6	69593ee5-976d-45d5-b66a-4f15cc9813ab	2	1	application/vnd.sas.decision.step.condition
14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef	50128647-a99b-4900-a7a1-15036f6f16ad	3	2	application/vnd.sas.decision.step.condition
4a989260-9cac-4e66-903a-f44f53a04721	0b45d0b3-5c99-4ce4-92bd-4662e8a2ebd2	4	3	application/vnd.sas.decision.step.condition
24dcb390-ae7d-4f85-8508-a97bdb486ee7	86b55dbb-47c0-4a5d-8789-479b02fb95f7	5	4	application/vnd.sas.decision.step.condition
95188ae9-74a9-404c-8904-12820d0f71f0	d9f1887e-e613-4667-b5d3-6dea3d006bd3	6	5	application/vnd.sas.decision.step.condition
ed3fc808-d148-409a-82cd-053edd9e181e	22f5be5d-aaf9-4a90-ab35-3658569cd48e	9	6	application/vnd.sas.decision.step.ruleset
8ed63501-73ed-4b67-aae0-40cb0b4ef986	22f5be5d-aaf9-4a90-ab35-3658569cd48e	10	6	application/vnd.sas.decision.step.condition
94e3112f-0876-466b-ad1f-b5b56097bb14	ef24817b-71d6-4c6c-be79-5678635c0712	16	7	application/vnd.sas.decision.step.condition
efbb7297-c27f-4f57-86b9-db9456356bb5	9d86dd1b-289e-42e1-b6f4-e5774c18c42e	17	8	application/vnd.sas.decision.step.condition
4e2fdfb4-da7a-4082-b96b-9b4eb167bab6	85acf56e-7324-4aac-8f03-ceed10297bde	19	9	application/vnd.sas.decision.step.ruleset

Display 5. Each Leg of the Path Traversal

INDIVIDUAL NODE COUNTS

Depending on the design of a decision, an individual node might be traversed via multiple paths. For many reasons, it is important to recognize which nodes in a decision are in heavy use or are in a critical path.

The `%dcm_nodes_counts` macro reads two tables:

- The `freqNodesTable=` parameter specifies the path frequency table produced by `%dcm_decision_path_frequency`
- The `pathNodestable=` parameter specifies the decision structure table produced by `%dcm_decision_path_nodes`

The output table has one row for each distinct node in the decision, with the following columns:

- The node type
- The node ID
- The node name
- The count, or the number of times that the node was executed during the decisions

The code in Example 6 uses data from the tables WORK.PATH_FREQ and WORK.PATH_NODES to produce the table WORK.NODES_COUNTS, shown in Display 6:

```
%dcm_decision_nodes_counts(URI=&URI, /* URI from prior EXECUTE example*/
    freqNodesTable=WORK.PATH_FREQ, pathNodesTable=WORK.PATH_NODES,
    outputTable=WORK.NODES_COUNTS)
```

Example 6. Generating Node Visit Count Data

counts	node_id	node_name	node_type
3202	14f0aa4e-6c23-4e5d-9aa2-50fa20d5ffef		Condition
1296	16037219-ebf2-4669-a22a-c65e8278b138		Condition
15	1755a970-1e47-4bcc-a743-792b20366795	HMEQ_expression_order	Ruleset
43	1820f3bd-1bb9-4308-9ef0-10da2100c5c2	HMEQ_loan_bin	Ruleset
2	1ae32dab-28d9-4dd3-9ec4-b35cd577b52e	HMEQ_value_NO_RULES	Ruleset

Display 6. Node Counts Table

Note that the output table from this macro does not include the nodes that were never traversed.

CONSIDERATIONS WHEN USING CAS DATA WITH THESE MACROS

The macros that we have presented have been shown using data in Base SAS libraries; however, the macros are also able to use data residing in CAS. All that is needed is for a SAS libref to be established using the CAS engine. Note that if the input table is read from a CAS libref, the output table must be written to a CAS libref that uses the same CAS session. The output table does not have to be written to the same CAS libref as long as it is accessed using the same CAS session.

When using CAS tables, an additional PROMOTE= **YES** | NO parameter is available to the macros to indicate whether the output CAS table should be promoted to global scope. The default is **YES**.

The code in Example 7 establishes a CAS session and executes same decision we have been working with, however this time the input data is read from the PUBLIC caslib and the output result table is written to the CASUSER caslib. The other macros would be similarly invoked.

```
cas mysess sessopts=(caslib=public);
libname public cas caslib=public;
libname casuser cas caslib=casuser;

%dcm_execute_decision(URI=&URI,
  inputTable=PUBLIC.hmeq_test, outputTable=CASUSER.decision_output);

libname casuser clear;
libname public clear;
cas mysess terminate;
```

Example 7. Executing a Decision in CAS

When using CAS data, the macros will run the underlying code in the CAS server. An indication that this is happening is by looking for the text shown in Output 1, indicating that the thread and or data program are running on "nodes".

```
NOTE: Running THREAD program on all nodes
NOTE: Running DATA program on all nodes

/* alternatively you may see the following */

NOTE: Running THREAD program on all nodes
NOTE: Running DATA program on one node
```

Output 1. SAS Log Messages Indicating Execution in CAS

Having the data in CAS makes it available for use with SAS Viya action sets, as well as additional SAS products such as SAS Visual Analytics and SAS Visual Data Mining and Machine Learning.

ANALYSIS EXAMPLES

Several questions come to mind when analyzing a decision flow; a few of these are:

- What nodes in the decision are "heavily" involved in the decision process?
- What is the distribution of paths utilized in reaching results of specific interest?
- What rules are involved with reaching results of specific interest?

These are these are just a few of the many questions that could be asked. We will use them as examples for demonstrating the use of SAS Visual Analytics with the data that we now have.

MOST ACTIVE NODES

Using SAS Visual Analytics, we can easily create a report of decision nodes that were most-actively visited, as shown in Figure 1. From visual inspection, we can tell which nodes are significantly involved in shaping the decision outcome. This information can be useful when considering modifications not only to the decision flow, but to the respective nodes.

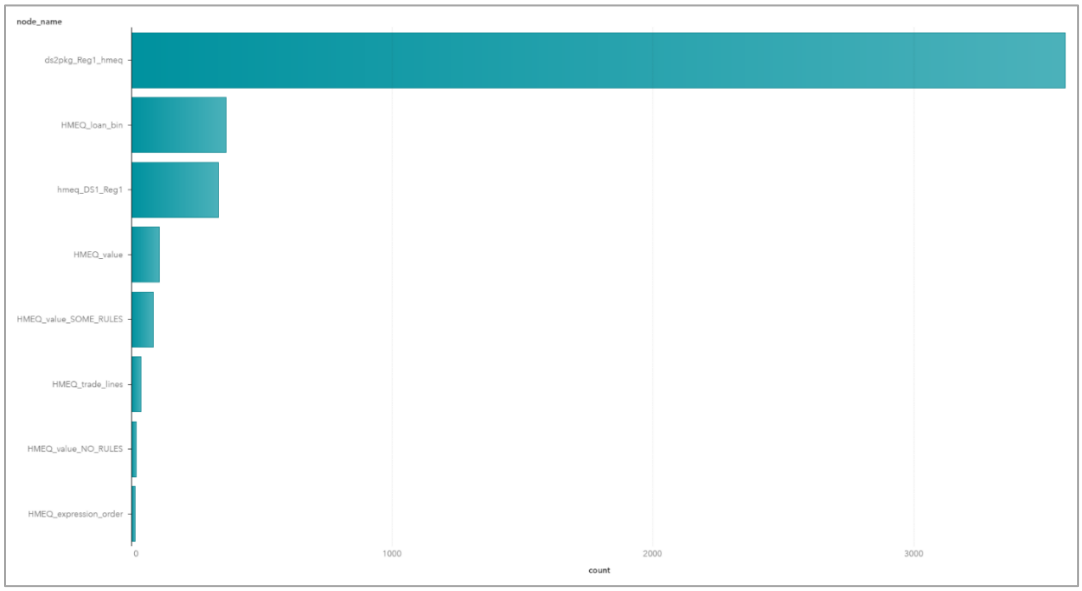
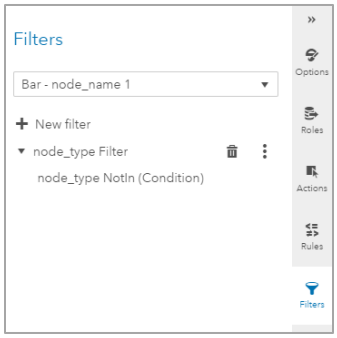


Figure 1. Most Actively Executed Nodes in a Decision

To build this report in SAS Visual Analytics, we select the ***NODES_COUNTS*** table as shown in Display 6 for our data source. We also select **NODE_NAME** as our category variable, and **COUNTS** as our measure. In addition, we define a filter on **NODE_TYPE**, specifying *NODE_TYPE NotIn(Condition),* as shown in Display 7. We do this because condition nodes are not assigned a name, and thus would otherwise all be lumped into a single bar titled "MISSING". This filter allows all other node types in the decision to show up in the bar chart by name.



Display 7. Filtering out Condition Nodes

OUTCOME RESULTS BY DECISION PATH

The example decision flow that we have been discussing processes loan request applications and has numerous logical paths. One of the output variables (**BAD**) in the example decision has values of 1 or 0, indicating whether the loan request should be approved. Other output variables provide specifics of the loan offer to extend to the customer.

Given that the results can be produced via numerous paths, a butterfly chart as shown in Figure 2 is useful to visualize the cumulative number of approvals or denials that are produced from each unique path through the decision.

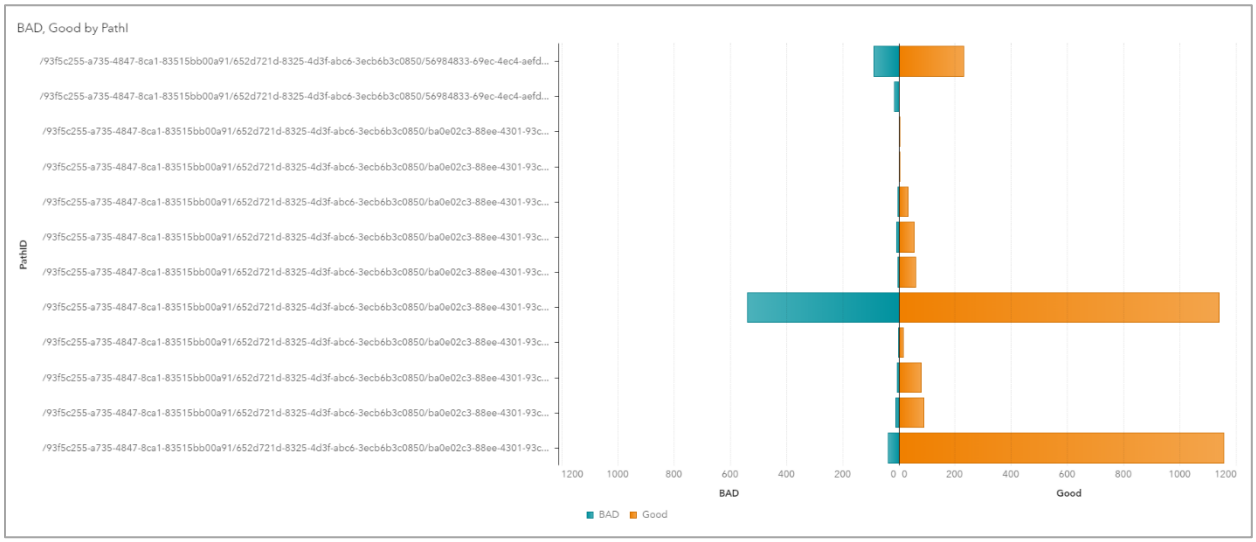
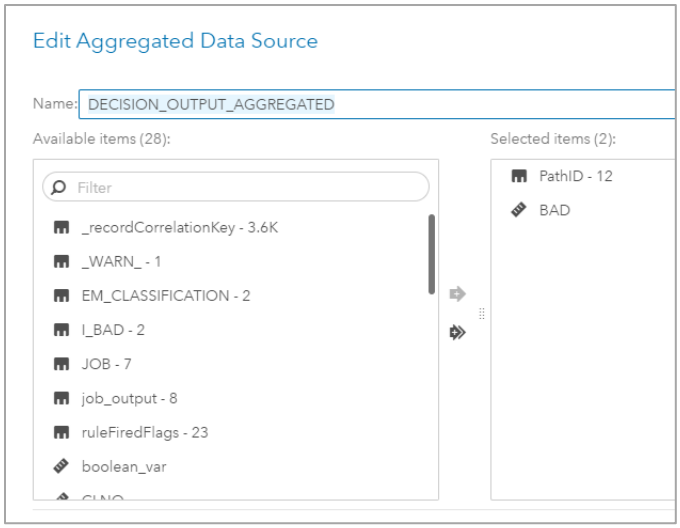


Figure 2. Decision Outcome Results by Decision Path Taken

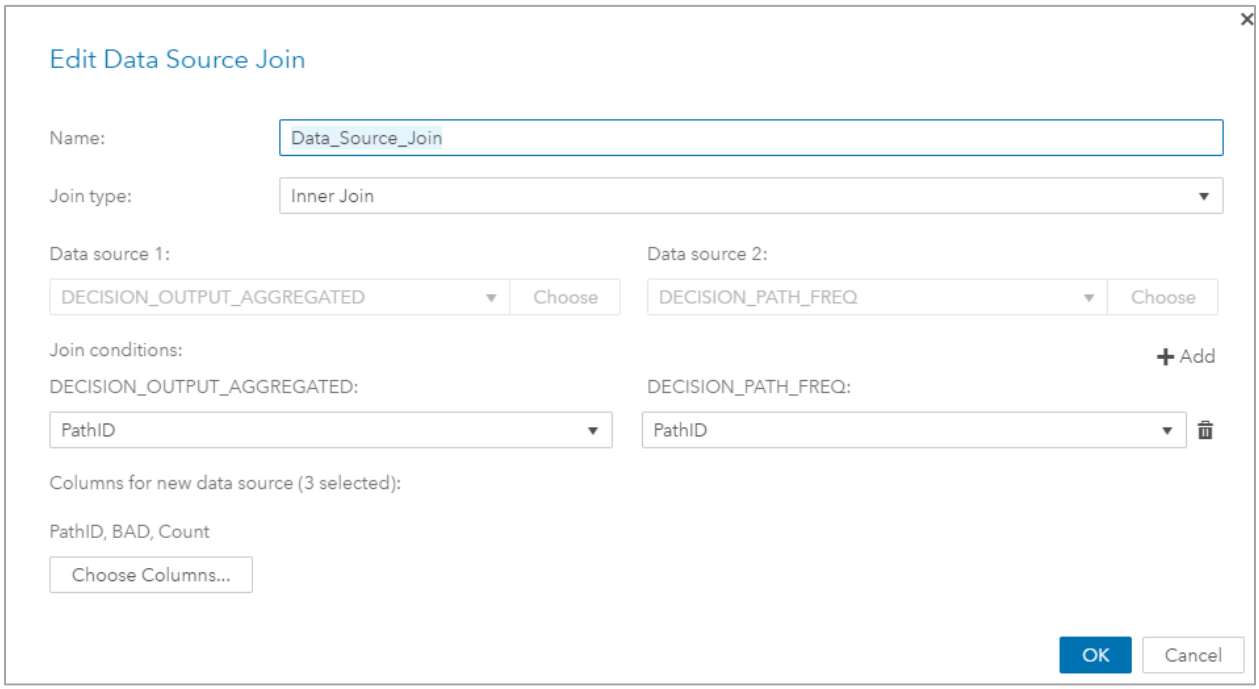
From a quick inspection, we can see the distribution of the results among the different paths. This visualization allows the analyst to investigate the performance of specific paths further because they might not be meeting expectations.

To build this report in SAS Visual Analytics, we need to enhance our available data sources. First, using the **DECISION_OUTPUT** table from Example 7, we create an aggregated data source named **DECISION_OUTPUT_AGGREGATED**, which summarizes the value of **BAD** by **PATHID**.



Display 8. Aggregated Data Source Definition

We then define a new joined data source, as shown in Display 9, with an inner join on **PathID**, the summed **BAD** column from **DECISION_OUTPUT_AGGREGATED**, and the **COUNT** column from **DECISION_PATH_FREQ**.



Display 9. Join Data Source Definition

To get the count of **GOOD** loan requests, we create a numeric calculated item as a measure in **Data_Source_Join** with the expression ***Count – BAD***.

We then use **Data_Source_Join** as the source of data for the butterfly chart. We assign **PathID** as the category variable, and **GOOD** and **BAD** as the two measure variables.

OUTCOME RESULTS AND RULES FIRED

The most heavily traversed path in Figure 2 has not only the most approvals, but also the most rejections. Although rejecting loan applications protects the organization from loan defaults, it is possible that loans are being rejected in cases where terms could be worked out with the applicants. If loans could be granted to these applicants with a low risk of default, the organization could realize more profit from a customer base that is being rejected at present.

To investigate this scenario, it is useful to know which rules were fired for the loans that were requested for this path. This requires joining the rejection records in **DECISION_OUTPUT** with the corresponding rule-fired detail records from **DECISION_RULEFIRE_DETAIL** for the most frequently traversed path that was identified from **DECISION_PATH_FREQ**.

The FedSQL code shown in Example 8 reveals a big surprise, as shown in Output 2: no rules were fired along this path. The default assumption of the decision is NOT to approve a loan request. However, the nodes along the decision path are designed to determine whether the application should be approved, as opposed to determining whether it should not be approved.

```
NOTE: Active Session now mysess.
NOTE: Added action set 'fedSql'.
NOTE: Table WHY_BAD was created in caslib CASUSER(brmdev) with 0 rows
```

Output 2. Query Results

```
cas mysess sessopts=(caslib=casuser);
options noquotelenmax;
proc cas;
  session mysess;
  fedSql.execDirect query="create table WHY_BAD {options replace=true} as
    select b.loan, b.job, b.mortdue, b.value, b.delinq, b.debtinc,
           c.rule_nm, c.rule_set_nm, c.rule_set_version_id

    from
(select PATHID from DECISION_PATH_FREQ order by COUNT desc limit 1) as a,
DECISION_OUTPUT         as b,
DECISION_RULEFIRE_DETAIL as c

where a.pathid = b.pathid
  and b.bad    = 1
 and  b._recordCorrelationKey = c._recordCorrelationKey;";
run; quit;
cas mysess terminate;
```

Example 8. Find All Rule-Fired Records for Specific Path Loan Rejections

This discovery changes the focus of the analysis. The question is now "What decision nodes are involved on this path, and why don't these applicant requests match the logic of these nodes, such that some might become approved?"

To answer these questions, we need to alter the query in Example 8 to find the rejection records for this path, regardless of rule-fired status. The modified query is shown in Example 9:

```
fedSql.execDirect query="create table WHY_BAD {options replace=true} as
       select b.loan, b.job, b.mortdue, b.value, b.delinq, b.debtinc
       from DECISION_OUTPUT as b
   (select PATHID from DECISION_PATH_FREQ order by COUNT desc limit 1) as a,
       where a.pathid = b.pathid and b.bad = 1;";
```

Example 9. Revised Query for Specific Decision Path Loan Rejections

This table can be used by the business analyst for further analysis. For example, the analyst can determine whether the decision should be revised to adapt to a customer segment that is identified by these records.

Along with this information, we need a query to report each node along the specific decision path. This will enable the business analyst to also examine this path to determine whether the business goals are being properly addressed. Here is an example:

```
fedSql.execDirect query="create table BAD_PATH {options replace=true} as
  select b.node_id as node_id, b.node_type as node_type,
         coalesce(b.model_nm,b.rule_set_nm,b.custom_object_nm,
                  'See Condition Node') as node_nm
  from (select PATHID from DECISION_PATH_FREQ
        order by COUNT desc limit 1) as a,
       DECISION_PATH_NODES as b
  where a.pathid = b.pathid
  order by b.node_depth_no, b.node_seq_no
  limit all;";
```

Example 10. Examining the Path of Interest

With respect to the query in Example 10, keep the following in mind:

- The COALESCE() function is used to produce a single column named **NODE_NM**. This is because the **DECISION_PATH_NODES** table has separate name columns based on the type of node.
- The literal text '*See Condition Node*' is used for node names of condition nodes. SAS Decision Manager does not currently support naming a condition node.
- The table is ordered by **NODE_DEPTH_NO** and **NODE_SEQ_NO** in order to show the actual traversal order of the decision.
- When using an ORDER BY clause in FedSQL in CAS, LIMIT ALL is needed.

The results of this query, shown in Table 2, suggest that the path that is taken starts with a model and then progresses through decision nodes. No other types of nodes are taken. Perhaps this was a path that had not yet been implemented, or that needs closer examination. In any case, the analyst now has a new opportunity to improve the business.

Obs	NODE_ID	NODE_TYPE	NODE_NM
1	93f5c255-a735-4847-8ca1-83515bb00a91	application/vnd.sas.decision.step.model	ds2pkg_Reg1_hmeq
2	652d721d-8325-4d3f-abc6-3ecb6b3c0850	application/vnd.sas.decision.step.condition	See Condition Node
3	ba0e02c3-88ee-4301-93c4-a05f91fe0d55	application/vnd.sas.decision.step.condition	See Condition Node
4	250aa1aa-dd52-48bc-b7bd-c86ac4ebb0b8	application/vnd.sas.decision.step.condition	See Condition Node
5	8f6c9efe-d188-4dbb-9b10-13dbf6cb1b3c	application/vnd.sas.decision.step.condition	See Condition Node
6	54a92d8f-26d1-4c08-b082-09e095fc9ace	application/vnd.sas.decision.step.condition	See Condition Node

Table 2. Path Traversal for High Rejection Rate

CONCLUSION

The addition of a macro interface to allow SAS programmers to explore all aspects of the data associated with rulesets and decisions enables them to leverage the full power of the SAS system to enhance and improve the quality and accuracy of their decisions. The ability to visualize how the paths and critical elements in their decisions creates a heightened awareness of how important specific nodes in the decision are to the overall structure.

This paper presents examples to enable SAS users to explore how they can do more with this data. For customers who are executing decisions in batch, traditional warehousing of the data presents numerous possibilities for refining decisions and evaluating their effectiveness.

REFERENCES

Crain, C. and C. Upton. 2016. "Taming the Rule." *Proceedings of the SAS Global Forum 2016*, Las Vegas, NV. Available http://support.sas.com/resources/papers/proceedings16/SAS6220-2016.pdf.

SAS Institute Inc. 2018. *SAS Visual Analytics 8.3: Working with Report Data.* Cary, NC: SAS Institute Inc. Available https://go.documentation.sas.com/?cdcId=vacdc&cdcVersion=8.3&docsetId=vareportdata&docsetTarget=titlepage.htm&locale=en

SAS Institute Inc. 2019. *SAS Visual Analytics 8.3: Working with Report Content.* Cary, NC: SAS Institute Inc. Available https://go.documentation.sas.com/?cdcId=vacdc&cdcVersion=8.3&docsetId=vaobj&docsetTarget=titlepage.htm&locale=en

SAS Institute Inc. 2018. *SAS Viya 3.4: FedSQL Programming for SAS Cloud Analytic Services*. Cary, NC: SAS Institute Inc. Available https://go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.4&docsetId=casfedsql&docsetTarget=titlepage.htm&locale=en

Example code. Available https://github.com/sascommunities/sas-global-forum-2019/tree/master/3473-2019-Sommer

ACKNOWLEDGMENTS

The author gratefully acknowledges the guidance and assistance of many SAS staff members, especially Chris Upton, Charlotte Crain, and David Wiehle.

CONTACT INFORMATION

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

Carl Sommer
SAS Institute Inc
Carl.Sommer@sas.com
www.sas.com