Paper SAS3030-2019

Data Quality Programming Techniques with SAS® Cloud Analytic Services

Nicolas Robert, SAS Institute Inc.

ABSTRACT

SAS® Cloud Analytic Services (CAS) provides advanced and powerful analytic capabilities on big data, both in-memory and in-parallel. But what do your reports, models, or decisions look like if your data is not of good quality? How do you improve the quality of big data by using distributed processing and in-memory techniques? CAS offers not only numerous actions for analyzing big data, but it also provides big data preparation and big data quality capabilities from a user interface perspective, as well as from a programming perspective. This paper focuses on the programming techniques that are available to perform common data quality tasks in parallel on in-memory (big) data.

INTRODUCTION

SAS® Data Quality programming capabilities in CAS are available through using traditional SAS® DATA step language elements or through using specific CAS actions. CAS actions are executable routines that the CAS server makes available to client programs. They are the smallest unit of work for the CAS server.

Those data quality capabilities are the exact same operations that are run under the covers when using the SAS® Data Preparation point-and-click user interface (SAS Data Explorer and SAS Data Studio). They provide the same results and run in the same technical environment, exploiting the CAS massively parallel processing architecture.

When you load a data table in a multi-machine CAS environment, the data is evenly distributed over the CAS workers. When you run an operation on that data, whether it is a statistical computation or a data transformation, each CAS worker processes the rows that are located on that node, making that operation much faster than equivalent processing on SMP environments. Data quality functions and actions leverage this powerful parallel architecture.

There are benefits in having data quality programming capabilities in addition to the graphical user interface: integration in a data flow, orchestration of data processing tasks, automation and scheduling of jobs, flexibility, and ease of customization.

This paper shows you how to perform the following data quality tasks programmatically:

- discover and profile data
- transform, cleanse, and enrich data
- cluster data based on matchcodes (fuzzy matching)
- de-duplicate data

DATA PROFILING

DATA DISCOVERY ACTION SET

In SAS® Viya®, you can profile data located in CAS using the *dataDiscovery.profile* CAS action. A data profiling task enables you to collect several metrics about your data: frequencies, basic statistics, patterns, outliers (highest and lowest values in a column), and

various other attributes. This helps you better understand the shape of your data and identify potential issues at a glance.

You can run a CAS action from any supported CAS client such as SAS, Python, R, Lua, Java, or the CAS REST APIs. In this paper, all the examples are run in SAS® Studio on SAS Viya.

Here is an example of the profile CAS action syntax:

```
proc cas ;
   dataDiscovery.profile /
      algorithm="PRIMARY"
      table={caslib="mydata" name="mycustomers"}
      columns={"id","name","address","city","state","zip"}
      cutoff=20
      frequencies=10
      outliers=5
      casOut={caslib="mydata" name="mycustomers_profiled"
              replace=true replication=0}
   ;
quit ;
```

The *profile* CAS action provides several options. You specify the input table and columns you want to profile, set some options to control the process (cutoff, frequencies, outliers, and so on), and name your output result table.

Display 1 is a sample display of the profiling output table.

ColumnId	RowId	Count	CharValue	DoubleValue
4	1000	0	state	.
4	1001	2		.
4	1002	55		.
4	1003	0	CharValue	.
4	1004	1	Va.	.
4	1004	1	Md.	.
4	1004	1	N.C.	.
4	1004	7	NC	.
4	1004	2	Maryland	.
4	1004	3	Virginia	.
4	1004	12	MD	.
4	1004	2	North Carolina	.
4	1004	26	VA	.

Display 1. Data Profiling Output Table

The output table gathers all the metrics collected in one single structure. Depending on your needs and on how you want to present the results, you might have to rework the output table.

The columns in the profiling output table are as follows:

- *ColumnId* designates the ID of the column in the source table that has been profiled.

- *RowId* designates the metric that the row contains (1000 is the column name, 1001 is the data type, 1002 is the count of values, 1003 is the name of the column containing the profiled column values, and so on). All the metrics IDs are described in the documentation.
- *Count* is a multi-purpose column which, depending on the metric in the row, might contain the metric's value (integer metrics).
- *CharValue* and *DoubleValue* are multi-purpose columns which, depending on the metric in the row, might contain the metric's value (character and numeric metrics).

The *dataDiscovery.profile* CAS action also offers identity analysis capabilities. This is useful to classify data strings according to specific rules and to determine what type of data an input string represents. For example, it can help to identify which fields contain personal information.

Identification analysis can be done using either a pattern defined as a regular expression or a SAS® Quality Knowledge Base (QKB) identity definition.

The QKB supplies rules and references data used to analyze and transform your data. For more information about the QKB, see the documentation (link in References).

Here is an example of identity analysis using a QKB identity definition:

```
proc cas ;
   dataDiscovery.profile /
      algorithm="PRIMARY"
      table={caslib="mydata" name="mycustomers"}
      multiIdentity=true
      locale="ENUSA"
      qkb="QKB CI 29"
      identities= {
         {pattern=".*", type="*", definition="Field Content",
          prefix="QKB_"}
      }
      cutoff=20
      frequencies=10
      outliers=5
      casOut={caslib="mydata" name="mycustomers_profiled" replace=true
              replication=0}
   ;
quit ;
```

In this example, we are interested in the following parameters:

- *qkb*: Specifies the name of the QKB deployed in CAS to be used in the identity analysis.
- *locale*: Specifies the language-country combination to be used in the identity analysis.
- *identities*: Specifies a list of identities to be used during identity analysis.
    - *pattern*: The regular expression to search for. When QKB identity analysis is in effect (which is the case in this example), the regular expression is applied to the output of the QKB identity analysis. ".*" means to "output the result as is".
    - *type*: The string that specifies the identified type. If set to "*", and "*definition*" is specified (which is the case in this example), it will be replaced by the actual string from QKB identity analysis output.
    - *definition*: The name of the QKB definition to be used for identity analysis.

- *prefix*: When QKB identity analysis is in use, and the type is set to "*", the prefix can be used to prefix the output string.

  In this example we are using a definition from the QKB called "Field Content". The Field Content definition is designed to analyze any input string and determine what type of data it represents.

- *multiIdentity*: Specifies how you want to interpret the identity analysis results.
  - *true* means that a weighted score will be assigned if the input value represents more than one identity.
  - *false* means that only a count of the most probable identity will be output.

Table 1 shows an example of using the *multiIdentity* option.

Input values	**multiIdentity=true**	**multiIdentity=false**
New York	QKB_CITY=106 QKB_STATE/PROVINCE=94	QKB_CITY=1
Washington	QKB_CITY=106 QKB_STATE/PROVINCE=94	QKB_CITY=1
California	QKB_STATE/PROVINCE=200	QKB_STATE/PROVINCE=1
Aggregated view	**QKB_STATE/PROVINCE=129.33** **QKB_CITY=70.66**	**QKB_CITY=2** **QKB_STATE/PROVINCE=1**

Table 1. Multi-identity Impact

You can see that depending on the multiIdentity option, the conclusion could be different:

- When multiIdentity=true, you could conclude that input values are most likely states/provinces.
- When multiIdentity=false, you could conclude that input values are most likely cities.

The *MultiIdentity=true* statement takes into account that a specified string could represent multiple identities and weights those identities accordingly.

As a conclusion on this topic, you might ask why you would want to run data profiling programmatically, since you can profile data in the SAS Data Preparation user interface? Here are a couple of reasons:

- Profiling big data can take a long time and a program could be easily scheduled.
- You might want to customize the profiling report and observe trends between different runs.
- The Data Preparation UI does not let you schedule your profiling jobs.

For more information about the *dataDiscovery.profile* CAS action and its metrics, see the documentation for the Data Discovery Action Set (link in References).

DATA QUALITY OPERATIONS

DATA STEP FUNCTIONS

SAS Viya provides functions to perform data quality tasks in parallel on distributed CAS tables. Those data quality functions are callable from the SAS DATA step language component which fully leverages CAS.

A SAS DATA step can of course be used from any SAS language client, but is also available as a CAS action, meaning that you can also run a DATA step in CAS from Python for example.

Table 2 lists all available SAS Data Quality functions that run in CAS in SAS Viya 3.4.

DQCASE	DQEXTINFOGET	DQEXTRACT	DQEXTTOKENGET
DQEXTTOKENPUT	DQGENDER	DQGENDERINFOGET	DQGENDERPARSED
DQIDENTIFY	DQIDENTIFYIDGET	DQIDENTIFYINFOGET	DQIDENTIFYMULTI
DQLOCALEGUESS	DQLOCALEINFOGET	DQLOCALEINFOLIST	DQLOCALESCORE
DQMATCH	DQMATCHINFOGET	DQMATCHPARSED	DQPARSE
DQPARSE CALL routine	DQPARSEINFOGET	DQPARSETOKENGET	DQPARSETOKENPUT
DQPATTERN	DQSTANDARDIZE	DQTOKEN	DQVER
DQVERQKB			

Table 2. SAS Data Quality Functions Supported in CAS

Here is an example of running common data quality functions on a CAS table:

```
data mylib.mycustomers_dq ;
   length gender $1 mcName mcAddress parsedValue
          tokenNames lastName firstName stateStd varchar(100) ;
   set mylib.mycustomers ;
   gender=dqGender(name,'Name','ENUSA') ;
   mcName=dqMatch(name,'Name',95,'ENUSA') ;
   mcAddress=dqMatch(address,'Address (Street Only)',95,'ENUSA') ;
   parsedValue=dqParse(name,'Name','ENUSA') ;
   tokenNames=dqParseInfoGet('Name','ENUSA') ;
   if _n_=1 then put tokenNames= ;
   lastName=dqParseTokenGet(parsedValue,'Family Name','Name','ENUSA') ;
   firstName=dqParseTokenGet(parsedValue,'Given Name','Name','ENUSA') ;
   stateStd=dqStandardize(state,'State/Province (Abbreviation)','ENUSA') ;
run ;
```

Since the input and output tables are CAS tables, this DATA step will run in CAS. Nothing is run on the SAS client, there is no data movement involved between CAS and the SAS client, and this DATA step will benefit from the massively parallel architecture of CAS.

The following data quality functions are used in this example:

- The *dqGender* function determines the gender from the name.
- The *dqMatch* function creates matchcodes based on the name and the address to be further used for clustering and de-duplication.
- The *dqParse* function returns a delimited string of name tokens.
- The *dqParseInfoGet* function is informative and returns the names of the tokens from the specified parsing definition.

The *put* statement writes the following information to the SAS log:

```
tokenNames=Prefix,Given Name,Middle Name,Family Name,Suffix,Title/Additional Info
```

- The two *dqParseTokenGet* functions extract specified tokens from the parsed value.

- The *dqStandardize* function standardizes states based on the “State/Province (Abbreviation)” definition.

Display 2 shows the output table from this example.

	gender	mcName	parsedValue	lastName	firstName
1	F	L@8L&Y8$$$$$$$$4#4&B$$$$$$$	/=/Susan/=//=/Woodward/=//=/	Woodward	Susan
2	M	MY7F$$$$$$$$$$C&B$$$$$$$$	/=/James/=//=/Briggs/=//=/	Briggs	James
3	F	L@8L&Y8$$$$$$$$4#4&B$$$$$$$	/=/Susan/=//=/Woodward/=//=/	Woodward	Susan
4	U	Y@B$$$$$$$$$$$4~&47$$$$$$$	/=/Stacey/=//=/Rhome/=//=/	Rhome	Stacey
5	F	FY7M$$$$$$$$$$&Y_B$$$$$$$	/=/Irene/=//=/Greaves/=//=/	Greaves	Irene
6	M	MY7F$$$$$$$$$$C&B$$$$$$$$	/=/James/=//=/Briggs/=//=/	Briggs	James
7	F	W7B8&B@8$$$$$$$3&~$$$$$$$$	/=/Kate/=//=/Lindamood/=//=/	Lindamood	Kate
8	M	C@B$$$$$$$$$$M@M$$$$$$$$	/=/Bob/=//=/Jones/=//=/	Jones	Bob
9	M	C@B4@B$$$$$$$$4~_M$$$$$$$	/=/Steve/=//=/Johnson/=//=/	Johnson	Steve
10	U	&M_Y4_B$$$$$$$&42W7$$$$$$$	/=/Ashley/=//=/Iversen/=//=/	Iversen	Ashley
11	M	W_4~_Y$$$$$$$$8&M78$$$$$$$	/=/David/=//=/Lester/=//=/	Lester	David
12	M	B@Y74$$$$$$$$$&B8Y_L$$$$$$	/=/Andrew/=//=/Morris/=//=/	Morris	Andrew
13	F	Y@B$$$$$$$$$$$4~&37$$$$$$$	/=/Stacy/=//=/Rhome/=//=/	Rhome	Stacy
14	M	B#B42$$$$$$$$$C_Y_B7$$$$$$	/=/Jeremy/=//=/Munsch/=//=/	Munsch	Jeremy
15	F	G7_W87BF$$$$$$$3Y74~7B$$$$$	/=/Christine/=//=/Fielding/=//=/	Fielding	Christine
16	F	G7_W87BF$$$$$$$3Y74~7B$$$$$	/=/Kristine/=//=/Fielding/=//=/	Fielding	Kristine

Display 2. Output Table Extract

Of course, this is just an example, but you can also run additional tasks:

- Analyze character patterns with the *dqPattern* function.
- Analyze identities (like profiling identity analysis) using the *dqIdentify* function.

The matchcodes generated in this example will be useful in determining if the table has potential duplicates using fuzzy matching.

DATA QUALITY CLUSTERING

ENTITY RESOLUTION ACTION SET

Once you have generated matchcodes that correspond to your business objects (individuals, products, and so on), you might want to cluster records, that is to group together records that share the same matchcode (or raw value) for one or more variables. Observations that are in the same cluster are good candidates for being potential duplicates.

The *entityRes.match* CAS action allows you to group records based on customized rules. For example, you can decide to cluster records of individuals if their name’s matchcode, their address’ matchcode, and their US state are strictly identical. This has the effect of doing fuzzy matching on the name and address columns with exact matching on the state column.

If records match based on the rule specified, those records will receive the same cluster ID value, that is the unique identifier of each cluster or group of related rows.

This is exactly what we want to do in the following example:

```
proc cas ;
   entityRes.match /
      clusterId="clusterID"
      inTable={caslib="mydata",name="mycustomers_dq"}
```

```
        matchRules={{
           rule={{
              columns={"mcName","mcAddress","stateStd"}
           }}
        }}
        nullValuesMatch=false
        emptyStringIsNull=true
        outTable={caslib="mydata",name="mycustomers_clustered",
                  replace=true} ;
quit ;
```

The *entityRes.match* CAS action does the following in this example:

- Reads the "mycustomers_dq" CAS table.
- Generates a cluster ID that is stored in the new variable "clusterID".
- Assigns the same cluster ID to multiple records when mcName (matchcode for name), mcAddress (matchcode for address), and stateStd (standardized state) are identical.
- Creates a new "mycustomers_clustered" CAS table to hold the results.

Display 3 shows four sample colored "multi-rows" clusters that group related rows together.

	clusterID	name	address	stateStd	mcName	mcAddress
1	AAAAAAAAAAAOAAAAAAAAAA==	Kristine Fielding	115 N Floyd Street	VA	G7_W87BF$$$$$$$3Y74~7B$$$$$	$$$$$$$$$$$$$
2	AAAAAAAAAAAOAAAAAAAAAA==	Christine Fielding	115 N Floyd St	VA	G7_W87BF$$$$$$$3Y74~7B$$$$$	$$$$$$$$$$$$$
3	AAAAAAAAAAAAAAAAAAAAAA==	Susan Woodward	152 Blackberry Ln	VA	L@8L&Y8$$$$$$$$4#4&B$$$$$$$	$$$$$$$$$$$$$
4	AAAAAAAAAAAAAAAAAAAAAA==	Susan Woodward	152 Blackberry Ln	VA	L@8L&Y8$$$$$$$$4#4&B$$$$$$$	$$$$$$$$$$$$$
5	AAAAAAAAAAAiAAAAAAAAAA==	April Lasser	5367 Rustic Elk Limits	MD	W&4_Y$$$$$$$$$$&MY7W$$$$$$$	$$$$$$$$$$$$$
6	AAAAAAAAAAAiAAAAAAAAAA==	April Lasser	5367 Rustic Elk Limits	MD	W&4_Y$$$$$$$$$$&MY7W$$$$$$$	$$$$$$$$$$$$$
7	AAAAAAAAAAABAAAAAAAAAA==	James Briggs	1507 Bear Springs Road	VA	MY7F$$$$$$$$$$C&B$$$$$$$$$	$$$$$$$$$$$$$
8	AAAAAAAAAAABAAAAAAAAAA==	James Brigs	1507 Bear Springs Rd	VA	MY7F$$$$$$$$$$C&B$$$$$$$$$	$$$$$$$$$$$$$
9	AAAAAAAAAAABAAAAAAAAAA==	James Briggs	1507 Bear Springs Rd	VA	MY7F$$$$$$$$$$C&B$$$$$$$$$	$$$$$$$$$$$$$

Display 3. Clustering Output

By "multi-rows" clusters, we mean clusters of records that potentially contain duplicates. "Single-row" clusters imply that there wasn't a match according to the rule specified.

You can specify multiple rules in the *match* CAS action. This will behave like an OR condition. For example, you can ask to group records of individuals if their name's matchcode, their address' matchcode, and their email OR phone number are identical. For this, you would specify the following condition:

```
        matchRules={
           {rule={{columns={"mcName","mcAddress","email"}}}},
           {rule={{columns={"mcName","mcAddress","phone"}}}}
        }
```

The *entityRes.match* CAS action is targeted to find potential duplicate records. It behaves well when matching rules are precise and clusters have small numbers of records. Using this CAS action to make large groups of records (like thousands of records) with general rules (like all "Male" or "Female") could result in poor performance.

DATA DE-DUPLICATION

DE-DUPLICATING DATA USING THE DATA STEP

After having clustered your data, you might want to de-duplicate related records. This can be done easily using the SAS DATA step and BY-group processing which fully exploit CAS parallelism. The DATA step runs in multiple threads on each node, one thread per BY group:

```
data mylib.mycustomers_dedup ;
   set mylib.mycustomers_clustered ;
   by clusterID updateDate ;
   if last.clusterID then output ;
run ;
```

This is a very simple de-duplication rule. For each "multi-rows" cluster, we want to keep and output the one with the most recent update value.

Obviously, you can specify more complex rules to build your own "Surviving Record Identification" mechanism.

DE-DUPLICATING DATA USING GROUPBYINFO

Another way to de-duplicate records is to use the *simple.groupByInfo* CAS action. This CAS action was introduced in SAS Viya 3.4 and allows not only to perform basic de-duplication, but also additional useful tasks related to BY-group processing.

The *groupByInfo* action helps you complete the following tasks:

- Identify BY-groups by assigning a numeric and consecutive group ID to the BY-groups.
- Get additional metadata on the BY-groups, like the group frequencies and the position of each record within the BY-group.
- Remove duplicate records.
- "Prototype data", that is, reduce the number of records inside a BY-group while keeping cardinality.

The *groupByInfo* action does not require computations on any numeric columns. Moreover, you can easily add fields to your output without requiring them to be part of the BY-groups.

Here is an example of using the *groupByInfo* CAS action for de-duplication:

```
proc cas ;
   simple.groupByInfo /
      table={caslib="mydata",name="mycustomers_clustered",
             groupBy={"clusterID"}}
      copyVars={"id","firstName","lastName","gender","address",
                "city","zip","stateStd","updateDate"}
      casOut={caslib="mydata",name="mycustomers_dedup",replace=true}
      position=1
      generatedColumns={"FREQUENCY","GROUPID","POSITION"}
      details=true ;
quit ;
```

This example simply outputs the first row (position=1) of the cluster (group by clusterID) to the target table. The CAS action provides options to add some metadata about the BY-groups, such as the number of records of each BY-group (FREQUENCY) and the position of the record in the BY-group (POSITION).

There are many other use cases for using the *groupByInfo* CAS action, such as sampling data with all the cardinality but with fewer records or reducing cardinality by removing groups that have less than a specified number of records.

FUTURE

SAS Data Quality capabilities will continue to be enriched in upcoming versions of SAS Viya.

In the next release, the CAS data quality functions available today in the DATA step will be available in DS2. Some Data Enrichment capabilities will be added for address verification and geocoding. Also, significant performance improvements will be made to both the *match* CAS action (better performance with big clusters) and the *groupByInfo* CAS action (better performance with high-cardinality by-groups).

Some survivorship capabilities (Survivor Record Identification mechanism to build the "golden record") are on the roadmap after next release.

CONCLUSION

SAS Viya provides data quality capabilities that leverage CAS parallel architecture. Using the programming techniques described in this paper, you can add great value to your data integration processes by profiling data, identifying data issues, remedying them using common data quality operations, clustering data, and de-duplicating records. This is essential to provide the tools to ensure that your data is cleansed and can be trusted as it is the foundation of your business reports, analytical models, and decisions.

REFERENCES

SAS Institute Inc. 2018. "Data Discovery Action Set: Syntax" in *SAS® Data Quality 3.4 / Data Quality Action Programming Guide.* Cary, NC: SAS Institute Inc. Available at https://go.documentation.sas.com/?cdcId=dqcdc&cdcVersion=3.4&docsetId=casactdq&docsetTarget=cas-datadiscovery-profile.htm&locale=en

SAS Institute Inc. 2018. "About SAS Quality Knowledge Base" in *SAS® Quality Knowledge Base for Contact Information 30.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/onlinedoc/qkb/30/QKBCI30/Help/qkb-help.html

SAS Institute Inc. 2018. "Functions Supported in CAS" in *SAS® Data Quality 3.4 / SAS Data Quality Language Reference.* Cary, NC: SAS Institute Inc. Available at https://go.documentation.sas.com/?cdcId=dqcdc&cdcVersion=3.4&docsetId=dqclref&docsetTarget=n04jgzd2t2lsr0n1phxrjq5y8bpp.htm&locale=en

SAS Institute Inc. 2018. "Entity Resolution Action Set: Syntax" in *SAS® Data Quality 3.4 / Data Quality Action Programming Guide.* Cary, NC: SAS Institute Inc. Available at https://go.documentation.sas.com/?cdcId=dqcdc&cdcVersion=3.4&docsetId=casactdq&docsetTarget=cas-entityres-match.htm&locale=en

SAS Institute Inc. 2018. "Simple Analytics Action Set: Syntax" in *SAS® 9.4 and SAS® Viya® 3.4 Programming Documentation / SAS Visual Analytics Programming Guide.* Cary, NC: SAS Institute Inc. Available at https://go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.4&docsetId=casanpg&docsetTarget=cas-simple-groupbyinfo.htm&locale=en

RECOMMENDED READING

- *Rineer, Brian 2018. "Doin' Data Quality in SAS® Viya®". Proceedings of the SAS Global Forum 2018 Conference. Cary, NC: SAS Institute Inc. Available at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2156-2018.pdf*

CONTACT INFORMATION

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

Nicolas Robert
SAS Institute Inc.
nicolas.robert@sas.com
http://www.sas.com