ABSTRACT
Python coders can now leverage the power of the SAS® Quality Knowledge Base and dramatically improve their data quality and data matching results. This session explores the capabilities now available to Python coders and gives coding examples and demonstrations showing how to leverage SAS Quality Knowledge Base capabilities such as parsing, standardization, and match-coding to better prepare data for analytics. Techniques for entity resolution and duplicate elimination are also explored.

INTRODUCTION
Data quality is a pervasive problem. There are some Python packages for data quality, but they are mostly about detecting or reporting on data quality, not for improving data quality. If you are using Python to improve data quality today, you are probably writing a lot of your own regular expressions and np.where() code.

For over twenty years SAS users have been able to use SAS® Data Quality and the SAS Quality Knowledge Base (QKB) to quickly and easily improve data quality, enrich data, and to facilitate duplicate elimination and entity resolution using fuzzy matching techniques.

Now, with SAS® Data Quality on SAS® Viya®, Python coders have access to the rules-based AI contained in the QKB and can leverage that same rich set of data quality capabilities.

Those capabilities include the following:
- Identification Analysis (to highlight data that is in the wrong place)
- Parsing (to take apart data into its constituent parts)
- Standardization (to correct inconsistent formatting)
- Gender Analysis (to get gender values from name information)
- Match Coding (to generate consistent codes for similar data values)
- And more! (Casing, Extraction, Pattern Analysis, Locale Guess, Language Guess)

The SAS QKB currently has support for 42 locales in 27 different languages, including languages using non-latin characters, such as Chinese, Japanese, Arabic, and Russian.

This paper will use a typical data quality problem and show you how the first five capabilities of the SAS QKB mentioned above can be leveraged from a Python/Jupyter notebook to transform data exhibiting a poor level of data quality into a higher level of data quality that you would require for downstream analytics.
EXAMPLE PROBLEM

The spreadsheet shown below contains the data that we will be using for our example.

![Spreadsheet Data That Exhibits Poor Data Quality](image)

**Figure 1.** Spreadsheet Data That Exhibits Poor Data Quality

This data contains multiple rows per individual. Say for the type of analysis we want to do with this data we would like to have just six rows, one for each person, with a total Amount per person, and the other data cleansed and consolidated. Also, we would like to have a Gender field (Male/Female/Unknown) added.

Here are the data quality challenges that we are facing:

- no unique identifier per individual.
- various formats of names, including nick-names and initials.
- various formats of phone and address information.
- missing data and incomplete data (e.g., some phone numbers missing area codes).
- data in the wrong places (e.g., Email in the Phone column).
- concatenated data that needs parsing (e.g., “Calgary AB T2T1T5” all in the City field).
- typos (“Toporowski” vs “Toperowski”, PostCode “G0L1TO” vs “GOL1T0”, etc.).

The simplified business rules that we will use for this example are that we would like to get the longest Name, Address, City, Phone, and Email for each individual, along with total Amount, standardized Province, PostCode, Phone, and a generated gender code, like this:

![Results of the Data Quality Processing That We Would Like to Achieve](image)

**Figure 2.** Results of the Data Quality Processing That We Would Like to Achieve
LOADING DATA INTO SAS® CLOUD ANALYTIC SERVICES

SAS Cloud Analytic Services (CAS) is the cloud-based server running on the SAS Viya high-performance, fault-tolerant analytics architecture. The smallest unit of work for the CAS server is a CAS action. CAS actions can load data, transform data, perform analytics, and create output.

To use CAS in Python and Jupyter notebook you need to make sure that you have the SAS Scripting Wrapper for Analytics Transfer (SWAT) installed. See https://github.com/sassoftware/python-swat for details.

From a Jupyter notebook we can then import the SAS SWAT package (import swat) and connect to SAS CAS (swat.CAS), and then set the active library with the setsessopt action.

```
In [1]: # import swat (SAS Scripting Language for Analytics Transfer),
# and pandas
import swat
import pandas as pd

In [2]: # Create a connection to CAS, specifying host name or url
# the SAS Viya server needs SAS Data Quality or SAS Data Preparation installed and QIK configured
viya_cas = swat.CAS(hostname='yourViyaHost',port=12345,username='yourUser',password='yourPw')
# Set an active library for this session
viya_cas.setsessopt('CasLib:Public')
```

Display 1. Setting Up the Environment and Connecting to CAS

Then we need to upload the data, in this case we used the read_excel method. To display the uploaded data, we can use CASTable, a swat DataFrame-like object:

```
In [3]: # to upload the excel file uncomment the line below. Not necessary if it's already uploaded, you can just use the existing excel
# testdata = viya_cas.read_excel('C:\Testdata\DataAccounts.xlsl', caslib=dict(name='CDN Accounts',caslib='Public', promote=True));
# use the swat CASTable object to treat a CAS Table like a pandas DataFrame
# testdata = viya_cas.CASTable('CDN Accounts')
testdata.head(10)
```

Display 2. Uploading Data and Then Fetching That Data Back for Display

```
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>Prov</th>
<th>PostCode</th>
<th>Amount</th>
<th>Phone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>030</td>
<td>Mr. Jacques Plante</td>
<td>14 Deris Road</td>
<td>Canley</td>
<td>QC</td>
<td>J3V 3A5</td>
<td>56.0</td>
<td>(613) 551-2374</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>Tony Sanduci</td>
<td>2129 31 Ave</td>
<td>Calgary</td>
<td>Alberta</td>
<td>T2Y 1T9</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>Anthony Sartucci</td>
<td>1514 66 Ave</td>
<td>Yarmouth</td>
<td>ON</td>
<td>L7G 3C1</td>
<td>400.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>Manish Pandhri</td>
<td>5160 10 Ave</td>
<td>Burlington</td>
<td>ON</td>
<td>L7L 1N1</td>
<td>300.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>Jack Stein</td>
<td>145 34 Ave</td>
<td>Calgary</td>
<td>Alberta</td>
<td>T2Z 1AQ</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>Mrs. Quinlan</td>
<td>131 45 Ave</td>
<td>Burlington</td>
<td>ON</td>
<td>L7L 3N7</td>
<td>200.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>Mr. Arnold Tophorski</td>
<td>38 Manhattan, Unit 1005</td>
<td>Ottawa</td>
<td>ON</td>
<td>KIZ 1E8</td>
<td>813 755-2313</td>
<td>100.0</td>
<td><a href="mailto:ArnoldTophorski@gmail.com">ArnoldTophorski@gmail.com</a></td>
</tr>
<tr>
<td>260</td>
<td>Ms. MJ Beal</td>
<td>4500 Sverker St E</td>
<td>Montreal</td>
<td>QC</td>
<td>H2Z 1E6</td>
<td>900.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>Mrs. K. Jackson</td>
<td>1450 55 Ave</td>
<td>Calgary</td>
<td>Alberta</td>
<td>T2Y 1T9</td>
<td>100.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Display 3. Output From the Display Data Action
DATA QUALITY AND DATA ENRICHMENT OPERATIONS

IDENTIFICATION ANALYSIS

The first data quality operation we will invoke is the `dqIdentify` function, through the `dataStep.runCode` CAS action. This inspects the City, Prov, PostCode, and Phone using the “Field Content” definition, in the English Canadian locale (ENCAN) of the QKB.

The results that come back show where these fields are empty, or where we are getting data different than we expected, using the newly created _Ident fields:

- The Phone_Ident field shows where Phone is empty or looks like an email.
- The Prov_Ident field shows where Prov is empty or looks like a phone number.
- The City_Ident field shows where the Prov Field is empty or contains Postal Code.
- The City_Ident field shows where the City fields contains more than just City.

```
In [4]: # run dataStep code invoking dq function to Identify contents of City, Province, PostCode fields
viya.dataStep.runCode(
    code=r'"dataStep public.testexcel_dq_from_Python ;
    City_Ident = dqIdentify(City,'Field Content','ENCAN');
    Prov_Ident = dqIdentify(Prov,'Field Content','ENCAN');
    Post_Ident = dqIdentify(PostCode,'Field Content','ENCAN');
    Phone_Ident = dqIdentify(Phone,'Field Content','ENCAN');
    run;"');
# Let's look at just the first six rows of our data ...
 dq = viya.CASTable('testexcel_dq_from_Python')
 dq.head(6)
```

Display 4. Commands and Output from the Identification Analysis

For a real-world problem, you might want to invoke the dqIdentify function on more fields, or even all fields, depending on the extent of the problems. In this case, we are only working on these four fields since we know from previous data profiling work on the spreadsheet that the problem of misplaced data is limited to these four fields.
RIGHT-FIELDING

Next, we want to use the intelligence gained with the use of the dqIdentify function within SAS DATA step code that is invoked with the dataStep.runCode CAS action.

In the simple example code below, we find those cases where Email, Phone and Postal Code are in the wrong spot, and move them to the correct spot (and update the _Ident indicator fields as well).

```
# using the above results of the SAS QCB Identity Analysis,
# run dataStep code to move stray phone, email, Postal code info to their correct places
via dataStep.runCode(
  code = '''
   data public.testexcel_dq_from_Python;
   if Phone_Ident = 'E-MAIL' then do;
      Email = Phone;
      Phone = ''; 
      Phone_Ident = 'EMPTY';
   end;
   if Post_Ident = 'PHONE' then do;
      Phone = PostCode;
      PostCode = '';
      Post_Ident = 'PHONE';
   end;
   if Prov_Ident = 'POSTAL CODE' then do;
      PostCode = Prov;
      Prov = '';
      Prov_Ident = 'POSTAL CODE';
   end;
   run;
'''
)
```

Display 5. The Code Used for Right-Fielding and the Results
PARSING

Next, we will use the `dqParse` and `dqParseTokenGet` functions to fix those rows where City, Province, and Postal Code have been concatenated together.

```
In [6]: %run dataStep code to combine City, Province, PostCode fields for problem rows and parse out correct info.
   'code': 'data public.testexcel_dq_2(drop-City_Ident Prov_Ident Post_Ident Phone_Ident parsedCPP);
   set public.testexcel_dq_2_from_python;
   if City_Ident = 'CITY' and (Prov_Ident='EMPTY' or Post_Ident='EMPTY') then do;
     parsedCPP = dqParse(CATX('-',City,Prov,PostCode), 'City - State/Province - Postal Code', 'ENCA');
   City   = dqParseTokenGet(parsedCPP, 'State/Province', 'City - State/Province - Postal Code', 'ENCA');
   Prov   = dqParseTokenGet(parsedCPP, 'State/Province', 'City - State/Province - Postal Code', 'ENCA');
   PostCode = dqParseTokenGet(parsedCPP, 'Postal Code', 'City - State/Province - Postal Code', 'ENCA');
   end;
   run;

   dq2 = vlyr.CASTable('testexcel_dq_2')
   dq2.to_frame()
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>City</th>
<th>Prov</th>
<th>PostCode</th>
<th>Amount</th>
<th>Phone</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2010</td>
<td>Mr. Jacques Plante</td>
<td>14 Dernis Road</td>
<td>Cantiy</td>
<td>QC</td>
<td>J2V 3J5</td>
<td>50.00</td>
<td>(609)-755-3347</td>
</tr>
<tr>
<td>1</td>
<td>2020</td>
<td>Tony Sarducci</td>
<td>2125 31 Ave</td>
<td>Calgary</td>
<td>Alberta</td>
<td>T2T 1T5</td>
<td>100.00</td>
<td><a href="mailto:tony.ouka@telus.ca">tony.ouka@telus.ca</a></td>
</tr>
<tr>
<td>2</td>
<td>2030</td>
<td>Anthony Sarducci</td>
<td>5084 Joel Avenue</td>
<td>Burlington</td>
<td>ON</td>
<td>L7W 7Y7</td>
<td>300.00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2040</td>
<td>Aman Singh</td>
<td>5084 Joel Avenue</td>
<td>Burlington</td>
<td>ON</td>
<td>L7W 7Y7</td>
<td>300.00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2050</td>
<td>Jack Plant</td>
<td>5084 Joel Avenue</td>
<td>Burlington</td>
<td>ON</td>
<td>L7W 7Y7</td>
<td>300.00</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2060</td>
<td>Mr. Arnold Toperovski</td>
<td>50 Metropol, Unit 1005</td>
<td>Ottawa</td>
<td>ON</td>
<td>K1Z 1E5</td>
<td>500.00</td>
<td>613-765-2313</td>
</tr>
<tr>
<td>6</td>
<td>2070</td>
<td>Ms MJ Belanger</td>
<td>4500 Sherbrooke St</td>
<td>Montreal</td>
<td>QC</td>
<td>H2Z 1E5</td>
<td>800.00</td>
<td>514-799-2309</td>
</tr>
<tr>
<td>7</td>
<td>2080</td>
<td>Mr. Anthony Sarducci</td>
<td>2125 31 Ave</td>
<td>Calgary</td>
<td>AB</td>
<td>T2T 1T5</td>
<td>100.00</td>
<td>403-205-7517</td>
</tr>
<tr>
<td>8</td>
<td>2090</td>
<td>Plant Jack</td>
<td>201-14 Denis Rd</td>
<td>Gatineau</td>
<td>QC</td>
<td>J8V 3J5</td>
<td>150.00</td>
<td>555-2334</td>
</tr>
<tr>
<td>9</td>
<td>2100</td>
<td>Mme Marie-Joelle Belanger</td>
<td>4500, rue Sherbrooke-Ouest</td>
<td>Montreal</td>
<td>PQ</td>
<td></td>
<td>900.00</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2110</td>
<td>Mme M J Belanger</td>
<td>4500 Sherbrooke O</td>
<td>Montréal</td>
<td>QC</td>
<td>H2Z 1E5</td>
<td>800.00</td>
<td>799-2329</td>
</tr>
<tr>
<td>11</td>
<td>2120</td>
<td>Jacques Plante</td>
<td>14, Chamarn Dam, ap 201</td>
<td>Cantiy</td>
<td>QC</td>
<td></td>
<td>40.00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2130</td>
<td>Aman Singh</td>
<td>5264 Joel Av</td>
<td>Burlington</td>
<td>ON</td>
<td>805-637-5119</td>
<td></td>
<td><a href="mailto:a.singh@otst.com">a.singh@otst.com</a></td>
</tr>
<tr>
<td>13</td>
<td>2140</td>
<td>Arnie Toperovski</td>
<td>30, privé Metropole, ap 1605</td>
<td>Ottawa</td>
<td>ON</td>
<td>K1Z 1E9</td>
<td>400.00</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2150</td>
<td>Jacques Plante</td>
<td>14 Denlee Unit 201</td>
<td>Cantiy</td>
<td>QC</td>
<td>J2V 3J5</td>
<td>10.00</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2160</td>
<td>JF Tremlay</td>
<td>P.O. Box 123</td>
<td>St-Marc-dû Lac-Lac</td>
<td>QC</td>
<td></td>
<td>200.00</td>
<td>819-555-4545</td>
</tr>
<tr>
<td>16</td>
<td>2170</td>
<td>Jean-Francois Tremlay</td>
<td>CP 123</td>
<td>Saint Marc</td>
<td>QC</td>
<td>G0L 1T0</td>
<td>90.00</td>
<td><a href="mailto:JF.Tremlay@bell.ca">JF.Tremlay@bell.ca</a></td>
</tr>
<tr>
<td>17</td>
<td>2180</td>
<td>Tremlay, JF</td>
<td>CP 123</td>
<td>St-Marc</td>
<td>Quebec</td>
<td>G0L 1T0</td>
<td>50.00</td>
<td>555-4545</td>
</tr>
<tr>
<td>18</td>
<td>2190</td>
<td>A. Toperovski</td>
<td>30 Metropol Private, Unit 1605</td>
<td>Ottawa</td>
<td>Ontario</td>
<td>K1Z 1E9</td>
<td>100.00</td>
<td>756-2123</td>
</tr>
<tr>
<td>19</td>
<td>2200</td>
<td>Tony Sarducci</td>
<td>2125 31 Ave</td>
<td>Calgary</td>
<td>AB</td>
<td></td>
<td>400.00</td>
<td><a href="mailto:tony.ouka@telus.ca">tony.ouka@telus.ca</a></td>
</tr>
</tbody>
</table>

Display 6. The Code Used for Parsing Apart the City, Province, and Postal Code, and the Results

Now that we have all information in the correct place, we can move on to standardizations, corrections, and enrichment.
STANDARDIZATION AND ENRICHMENT

We want to standardize the Prov, PostCode, and Phone fields using the dqStandardize function. You want to use the correct standardization definition on each data type. (for example, the “State/Province (Postal Standard)” definition on the Prov field). We also specify the locale as “ENCAN”, so that the data gets standardize to “English, Canadian” standards (note that the ENCAN and FRCAN definitions handle data in both French and English). If we had some USA data, we would want to use the “ENUSA” definition on those rows.

Here we also invoke the dqGender function to generate the gender field. It will get a value of M, F, or U, depending on what it finds in the Name field. It considers name prefixes (Mr., Ms, Mme, etc.) and uses a lookup table of given names that skew toward a specific gender.

```
%let data = "sas.xls";
%let locale = "ENCAN";
%let standardize = "dqStandardize(Prov,'State/Province (Postal Standard)',"ENCAN");
%let gender = dqGender(Name,'Name','ENCAN');
```

Display 7. Standardization of Prov, Phone, and PostCode, and Generating Gender
ENTITY RESOLUTION OPERATIONS

MATCH-CODING

Next is the dqMatch function. Sensitivity is the third parameter, which determines how much “fuzziness” is allowed in the matchcode. The standard setting for sensitivity is 85, but can be set higher to require closer matches, or set lower to permit “fuzzier” matches. Both Canadian locales (ENCAN and FRCAN) will handle bilingual English and French data and generate the same match code for similar data no matter the language. If we had some USA data, we would probably want to use the “ENUSA” definition on those rows.

Display 8. Similar Data Values Get the Same Matchcode Using the dqMatch Function
CLUSTERING

Next we need to load the Entity Resolution CAS action set, and use the `entityres.match` CAS action to cluster together records based on the matching rules that we specify. In this example, we are using the matchcodes to match on the following:

- Name & Street Address & PostalCode - OR -
- Name & City & Province - OR -
- Name & Phone (last 7 digits) - OR -
- Name & Email

```r
# Load Entity Resolution CAS action set
viya.loadactionset(actionset="entityRes")
```

**Note:** Added action set 'entityRes'.

```r
# use the entityres.match CAS action to match rows on:
# (Name & Address & Postal Code) OR (Name & City & Province) OR (Name & Phone) OR (Name & Email)

dq_clustered = viya.CASTable('test_clustered', groupBy='CLUSTERID', replace=True)
viya.entityres.match(clusterID='CLUSTERID', intable=dq,
    match_rules='[rule: [columns: ['Name_matchcode55', 'Address_matchcode70', 'Postalcode']],
    [rule: [columns: ['Name_matchcode55', 'City_matchcode85', 'Province']],
    [rule: [columns: ['Name_matchcode55', 'Phone']],
    [rule: [columns: ['Name_matchcode55', 'Email']]]]
    outtable=dq_clustered)
```

```r
# display cluster results. CLUSTERID is a 24-byte character string
dq_clustered[['CLUSTERID', 'ID', 'Name', 'Address', 'City', 'Phone', 'Email']].sort_values('CLUSTERID').to_frame()
```

**Display 9. Clustering Rules Using Matchcodes, and the Resulting CLUSTERID**

The result of the entityres.match CAS action is a new column, which is called CLUSTERID here. All rows that fall into the same cluster, according to our matching rules, will get the same CLUSTERID value. This column is a 24-byte character string. Minor differences in the CLUSTERID values are a little difficult for most humans to detect. Therefore, you might want to transform the CLUSTERID into a numeric value.
Here we use the simple.groupByInfo CAS action to turn the CLUSTERID 24-byte character string into a numeric value called _GroupID_. Now all rows in the same cluster also have the same numeric _GroupID_ value.

Display 10. Transforming the 24-byte CLUSTERID Into a Numeric _GroupID_

You might have noticed that the QKB was not involved in this clustering step. So you could, if you wanted to, download the data from the server and do the clustering on your client, using Python code instead of using the entityres.match CAS action on the server. The advantages of doing it on the server are better performance and reduced network traffic. This doesn't really matter when dealing with just 20 records, but it certainly would matter if we were dealing with 20 million records.

Now that we have all the records for our six people grouped together as we had hoped we would, all that’s left to do is sum the amounts for each person, and choose the best information that we have for each person on each of the other columns. This is called Survivorship.
SURVIVORSHIP

Below is some example SAS DATA step code to create a surviving record for each person. In this example, we are simply summing up the Amount for each cluster, and selecting the Name, Address, City, Postal, Phone, and Email with the longest string lengths in the cluster. Real world survivorship rules are usually more robust than this, so please don’t consider this code to be a “best-practices” survivorship coding example.

Again, this step could be done with Python code on the client side. The advantages of doing it on the server are better performance and reduced network traffic. After this step is done, you only need to download one record per individual to your client. Or leave the data on the server and leverage SAS Analytics in CAS from Python as well!

```
# run dataset code to create golden record for each cluster with best info, and total amount

vliya.dataStep.runCode(
  code='''
  data public.test_done;
  (drop= max_Name min_gender max_Address max_City max_Postal tot_Amount max_Phone max_Email
    CLUSTERID ID Name_matchcode@ Address_matchcode@ City_matchcode@ Postal_code@ Phone?);
  set public.test_clust_nums;
  by _GROUPID;"

  retain max_Name min_gender max_Address max_City max_Postal tot_Amount max_Phone max_Email;
  if first._GROUPID then do;
    max_Name = Name;
    min_gender = gender;
    max_Address = Address;
    max_City = City;
    max_Postal = Postal;
    tot_Amount = Amount;
    max_Phone = Phone;
    max_Email = Email;
  end;
  else do;
    if length(Name) < length(max_Name) then Name = max_Name;
    if gender > min_gender then gender = min_gender;
    if length(Address) < length(max_Address) then Address = max_Address;
    if length(City) < length(max_City) then City = max_City;
    if length(Postal) < length(max_Postal) then Postal = max_Postal;
    tot_Amount = Amount + tot_Amount;
    if length(Phone) < length(max_Phone) then Phone = max_Phone;
    if length(Email) < length(max_Email) then Email = max_Email;
  end;
  if last._GROUPID then output;
run;'')

dq_DONE = vliya.CASTable('test_done');
dq_DONE[['_GROUPID', 'Name', 'gender', 'Address', 'City', 'Prov', 'Postal', 'Amount', 'Phone', 'Email']].sort_values('_GROUPID').

```

Display 11. Surviving Just One Record Per Individual
CONCLUSION

This paper has shown that the rules-based AI capabilities in the SAS QKB are a powerful way for you, as a Python coder, to achieve better data quality more quickly and easily than you could by trying to write your own code to achieve the same results.

What’s next? I recommend you try out these capabilities on a data quality problem of your own, or take the example shown here and take it further. What if you wanted to standardize the formatting of the Names and remove name prefixes, or add missing name prefixes? (hint: there are Name Standardization and Name Parsing definitions in the QKB).

Finally, you might be starting to think about how to move beyond just an interactive data science session in Jupyter and thinking about how to operationalize this data quality process. Will you want to schedule this as a regular batch job? Or turn it into a web service, callable in real-time by other applications? Or embed it in a process ingesting streaming data? All these deployment options for data quality are available with SAS.

ACKNOWLEDGMENTS

Thanks go out to Kevin D. Smith for his guidance on coding style, and to colleagues Ron Yee, André Lafreniere, and Guy Bourassa for content suggestions for this paper.

RECOMMENDED READING

- **SAS® Blogs**: "Using Python to work with SAS Viya and CAS", by Chris Hemedinger
- **SAS® Viya®: The Python Perspective**, by Kevin D. Smith and Xiangxiang Meng
- **SAS® Data Quality: Getting Started**
- **SAS® Quality Knowledge Base for Contact Information: Online Help**
- **SAS® Data Quality: Language Reference** (the “Functions supported in CAS” section)
- **SAS® Viya®: System Programming Guide** (the Python syntax examples)
- **SAS® Data Quality: CAS Action Programming Guide**
- **SAS® Cloud Analytics Services: CASL Programmer’s Guide**

CONTACT INFORMATION

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

Arnold Toporowski
SAS Institute (Canada) Inc
+1 (613) 755-2313
Arnold.Toporowski@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.