SESUG Paper 224-2018

Abstracting and Automating Hierarchical Data Models: Leveraging the SAS® FORMAT Procedure CNTLIN Option To Build Dynamic Formats That Clean, Convert, and Categorize Data

Troy Martin Hughes

ABSTRACT

The SAS® FORMAT procedure "creates user-specified formats and informats for variables."[i] In other words, FORMAT defines data models that transform (and sometimes bin) prescribed values (or value ranges, in the case of numeric data) into new values. SAS formats facilitate multiple objectives of data governance, including data cleaning, the identification of outliers or new values, entity resolution, and data visualization, and can even be used to query or join lookup tables. SAS formats are often hardcoded into SAS software, but where data models are fluid, formats are best defined within control files outside of software. This modularity—the separation of data models from the programs that utilize them—allows SAS developers to build and maintain SAS software independently while domain subject matter experts (SMEs) separately build and maintain the underlying data models. Independent data models also facilitate master data management (MDM) and software interoperability, allowing a data model to be maintained as a single instance, albeit implemented not only with SAS but also Python, R, or other languages or applications. The CNTLIN option (within the SAS FORMAT procedure) facilitates this modularity by creating SAS formats from data sets. This text introduces the BUILD_FORMAT macro that greatly expands the utility of CNTLIN, allowing it to build formats not only from one-to-one and many-to-one format mappings but also from multitiered, hierarchical data models that are built and maintained externally in XML files. The numerous advantages of BUILD_FORMAT are demonstrated through successive SAS code examples that rely on the taxonomy of the Diagnostic and Statistical Manual of Mental Disorders, Fifth Edition (DSM-5).

INTRODUCTION TO SAS FORMATS

The Diagnostic and Statistical Manual (DSM) of Mental Disorders, in addition to the International Classification of Diseases (ICD), is the authoritative source for classifying mental health disorders, with the fifth edition (DSM-5) updated in 2017.[ii] The DSM-5 is relied upon by psychiatrists, psychologists, and other clinicians for diagnosing and discriminating among disorders, and by healthcare insurers, billing departments, and other stakeholders for classifying metadata associated with healthcare. Table 1 provides a *very* abridged list of diagnoses and their respective DSM-5 and ICD-10 codes.

DSM-5 Code	ICD-10 Code	Disorder Name
291.0	F10.221	Alcohol dependence with intoxication delirium
291.1	F10.26	Alcohol dependence with alcohol-induced persisting amnestic disorder
291.2	F10.27	Alcohol dependence with alcohol-induced persisting dementia

Table 1. Abbreviated List of Equivalent DSM-5 and ICD-10 Codes

The most basic use of SAS formats is to alter the way that data are displayed—by temporarily or permanently transforming the data. For example, a data set might contain DSM-5 codes, but a physician might need to display the disorder name instead within a report. Rather than having to maintain both the disorder name and respective DSM-5 codes for all diagnoses, the FORMAT procedure can be used to transform the DSM-5 code into the associated disorder name.

To simulate this use case, the following code creates and prints a sample data set (Codes) that contains diagnostic data, with DSM-5 codes listed in the Dsmcode variable:

```
data codes;
    length dsmcode $8;
    label dsmcode='DSM-5 Code';
```

```
    dsmcode='291.0'; output;
    dsmcode='291.0'; output;
    dsmcode='291.1'; output;
    dsmcode='17'; output;
run;

proc print data=codes noobs label;
run;
```

This code produces the following output, printing the entire data set:

DSM-5 Code
291.0
291.0
291.1
17

However, to display the disorder name rather than its respective DSM-5 code, a SAS format—let's call it *DSM*—can be created and subsequently applied during the PRINT procedure. Thus, the following code temporarily transforms the display of the Dsmcode variable while the underlying data remains unchanged:

```
proc format;
    value $ dsm
    '291.0'='Alcohol dependence with intoxication'
    '291.1'='Alcohol dependence with alcohol-induced persisting amnestic disorder'
    '291.2'='Alcohol dependence with alcohol-induced persisting dementia';
run;
proc print data=codes noobs label;
    format dsmcode $dsm.;
run;
```

The code produces the following output, now displaying the disorder name rather than the DSM-5 code:

DSM-5 Code
Alcohol dependence with intoxication
Alcohol dependence with intoxication
Alcohol dependence with alcohol-induced persisting amnestic disorder
17

Note that because "17" lies outside of the data model (i.e., the DSM format defined by the previous FORMAT procedure), its value is unchanged after the format is applied.

This example demonstrates a one-to-one correlation between unformatted values and formatted values, because one disorder code references one disorder name, representing an abstraction of the disorder itself. Formats are also used to standardize, bin, and group values in a many-to-one relationship in which many unformatted values are transformed into a single formatted value. For example, if an analyst wanted instead to group all alcohol-related disorders (and wasn't interested in identifying any *specific* alcohol-related disorder), the following FORMAT procedure would create the desired format (DSMBIN) and resultant binning:

```
proc format;
    value $ dsmbin
    '291.0','291.1','291.2'='Alcohol-related disorder';
run;
proc print data=codes noobs label;
    format dsmcode $dsmbin.;
run;
```

The code creates the DSMBIN format and produces the following output:

DSM-5 Code
Alcohol-related disorder
Alcohol-related disorder
Alcohol-related disorder
17

The FREQ procedure also demonstrates the importance of binning values with SAS formats, as a single underlying data set can be maintained while different—even conflicting—formats are applied in separate analyses or by different stakeholders. The following code performs three frequency analyses on the Codes data: 1) applying no format, 2) applying the one-to-one DSM format, and 3) applying the many-to-one DSMBIN format:

```
proc freq data=codes;
    tables dsmcode/nopct nocum norow;
run;
proc freq data=codes;
    tables dsmcode/nopct nocum norow;
    format dsmcode $dsm.;
run;
proc freq data=codes;
    tables dsmcode/nopct nocum norow;
    format dsmcode $dsmbin.;
run;
```

The respective frequency tables are demonstrated in Table 2.

dsmcode	Frequency
17	1
291.0	2
291.1	1

dsmcode	Frequency
17	1
Alcohol dependence with intoxication	2
Alcohol dependence with alcohol-induced persisting amnestic disorder	1

dsmcode	Frequency
17	1
Alcohol-related disorders	3

Table 2. The Effect of SAS Formats on the FREQ Procedure

The previous two examples did not modify underlying data but only changed how the data are displayed. However, SAS formats can also be used to transform data permanently. One method is to use the PUT statement, which the following code demonstrates by creating the Dsmname variable that contains the disorder name:

```
data codeswithname;
    set codes;
    length dsmname $100;
    label dsmname='DSM-5 Name';
    dsmname=put(dsmcode,$dsm.);
run;
```

However, as the LENGTH statement demonstrates, significantly greater file sizes can occur where lengthy character values (e.g., Dsmname) are maintained as text rather than as formatted values. Thus, one of the principle advantages of appropriately implementing SAS formats is file size reduction, as well as the corresponding reduction in system resource utilization (i.e., CPU cycles, input/output calls, memory) and increased performance due to processing smaller data sets.

It goes without saying that a pesky “17” has pervaded the previous examples due to some dirty data in the Codes data set. The erroneous value falls outside of the data models (represented by the SAS formats DSM and DSMBIN), which segues into how formats can also be used to clean data. In some cases, it may be necessary to identify and group alternative or incorrect spellings to transform them into a correct, standardized value. For example, if a patient data set contained a variable Race that indicated patient race, analysts might have observed the invalid values “white”, “cauc”, and “caucasian” and might need to standardize them to the value “Caucasian”. This data quality binning is often hardcoded through conditional logic such as the following DATA step:

```
data cleanpatient;
    set dirtypatient;
    if lowcase(race) in ('white','cauc','caucasian') then race='Caucasian';
run;
```

However, identical quality controls can be implemented by creating a SAS format that is used to identify known value variations and to transform them into standardized values:

```
proc format;
    value $ race
    'white','cauc','caucasian'='Caucasian';
run;
```

This type of data quality binning requires knowledge of the invalid values that will be encountered, just as binning multiple disorders into the “Alcohol-related disorders” category requires knowledge of the DSM-5 code for each specific disorder. In other cases, the invalid (or undesirable) values should not be binned to a value but rather should be deleted, expunged, excluded, or relegated to an exception report. The OTHER option (when implemented with the VALUE statement of the FORMAT procedure) bins all unspecified values into a single value and offers an effective way to identify and handle rogue data during data cleaning operations. The use of SAS formats to further data governance objectives—including the use of the OTHER option—is not further discussed in this text but is an important use case of formats that should not be overlooked.

SOME SAS FORMAT LIMITATIONS

Although SAS formats excellently perform one-to-one transformations (e.g., DSM code to disorder name, or vice versa) and many-to-one transformations (e.g., DSM code to disorder group), SAS formats are limited in their ability to represent more complex data models:

1. SAS formats handle one-to-one relationships between entities well, but because formats are unidirectional, this is better represented as *ONE-to-one* relationship, not *one-to-one*. Thus, a separate format would be required to reverse this direction, essentially the *one-to-ONE* version.
2. SAS formats do not handle one-to-one-to-one relationships well. For example, given three equivalent abstractions for a disorder (e.g., disorder name, DSM-5 code, and ICD-10 code), six potential transformations exist: name to DSM-5 code, name to ICD-10 code, DSM-5 code to name, DSM-5 code to ICD-10 code, ICD-10 code to name, and ICD-10 code to DSM-5 code. Thus, were a data set to use both DSM-5 and ICD-10 codes, separate formats would need to be constructed for each transformation type. *Because DSM-5 and ICD-10 codes describe the same knowledge domain (i.e., mental health disorders), a best practice would be to maintain a single data model in which both DSM-5 and ICD-10 codes were included.*
3. SAS formats can represent one-to-one relationships or many-to-one relationships, but they can’t do both at once, nor can they represent multiple tiers of many-to-one relationships in which data are binned into a hierarchical taxonomy. For example, the DSM-5 code 291.0 reflects both an “Alcohol dependence with intoxication” disorder name and an “Alcohol-related disorder” classification; however, two FORMAT procedures are required to create these two views of the disorder, and an additional transformation (and FORMAT procedure) would be required to classify this disorder further in the “Substance use and addictive disorders” category that include both alcohol- and non-alcohol-related substance-related disorders. This complexity quickly becomes cumbersome because modifications to a single disorder require updates to all related FORMAT procedures. *A best practice is to maintain a single, hierarchical data model that can be used to*

reflect specific values at any level throughout the model. Where applicable, this model should also include abbreviations, acronyms, common misspellings, and other spelling variations.

4. SAS data models defined in SAS software are limited not only in the way they can represent data (as demonstrated by the previous three limitations), but also in their interoperability among other applications and the ease with which the data models can be updated. For example, a clinician might painstakingly encode the entire DSM-IV into a series of FORMAT procedures, but by the time this is completed, the DSM-5 might have been released, requiring painful, redundant updates to multiple FORMAT procedures. This method lacks maintainability and reusability, and moreover, coworkers who instead develop in Python or R would not be able to benefit from the hardcoded data model encoded in SAS FORMAT procedures. *A best practice is to maintain complex or dynamic data models outside of SAS—or any language or application—so they can be modified and maintained more readily while providing utility to the largest possible user base. This modularity also enables software experts to maintain software while knowledge domain experts maintain domain data models.*

The remaining sections demonstrate how to overcome these FORMAT challenges by implementing external data models that are more accurate abstractions of their real-world constructs. This data-driven paradigm and design shift supports master data management (MDM) objectives, as well as software maintainability, modularity, reusability, and interoperability.

MODULARITY: EXTERNALIZING SAS FORMATS WITH XML

The International Organization for Standardization (ISO) defines modularity as "the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components."[iii] Software modularity is often conceptualized as separating software into chunks; but modularity also includes the concept that data models, business rules, and other sources of program control should be separated from the software that processes them. This modularization enables domain SMEs to incrementally build, modify, and maintain data models (including disparate, conflicting, or incomplete data models) while software developers separately build, modify, and maintain the underlying software. Competing external data models are introduced and demonstrated in the author's first-ever SAS-related text: *Binning Bombs When You're Not a Bomb Maker: A Code-Free Methodology to Standardize, Categorize, and Denormalize Categorical Data Through Taxonomical Control Tables*.[iv] Software modularity and the benefits thereof are thoroughly discussed throughout the author's text: *SAS® Data Analytic Development: Dimensions of Software Quality*.[v]

To demonstrate the benefits of externalizing SAS formats, a slightly more complex data model is required, one which demonstrates more clearly the taxonomy and hierarchical nature of DSM-5 disorders. Table 3 is an abridged yet more complex representation of the DSM-5.

Classification 1	Classification 2	Disorder Name	DSM-5 Code
Substance use and addictive disorders	Alcohol-related disorders	Alcohol dependence with intoxication delirium	291.0
		Alcohol dependence with alcohol-induced sleep disorder	291.82
		Unspecified alcohol-related disorder	291.89
	Substance-related disorders	Substance dependence with intoxication delirium	292.81
		Stimulant use disorder	304.40

		Other or unknown substance-related disorder	304.90

Table 3. Abridged DSM-5 Disorder Classifications, Names, and Codes

In this example, the Extensible Markup Language (XML) is used to represent the DSM-5 data model. For those unfamiliar with XML or how XML is ingested and interpreted by SAS programs, XML can be reverse engineered by constructing a data model as a SAS data set, after which it is exported to XML. This method demonstrates the XML structure, format, characteristics, and content that SAS creates, and which it in turn expects when ingesting XML files created outside of SAS. In addition to XML, other file formats such as Excel or raw text can be used to abstract data models for ingestion into SAS; however, only XML data models are demonstrated in this text.

The following code builds the contents of Table 3 as the DSMmodel data set:

```
data DSMmodel;
   infile datalines delimiter=',';
   length class1 $50 class2 $50 dsm5name $100 dsm5code $8;
   input class1 $ class2 $ dsm5name $ dsm5code $;
   datalines;
Substance use and addictive disorders,Alcohol-related disorders,Alcohol dependence
with intoxication delirium,291.0
Substance use and addictive disorders,Alcohol-related disorders,Alcohol dependence
with alcohol-induced sleep disorder,291.81
Substance use and addictive disorders,Alcohol-related disorders,Unspecified alcohol-
related disorder,291.89
Substance use and addictive disorders,Substance-related disorders,Substance
dependence with intoxication delirium,292.81
Substance use and addictive disorders,Substance-related disorders,Stimulant use
disorder,304.40
Substance use and addictive disorders,Substance-related disorders,Other or unknown
substance-related disorder,304.90
;
```

This code would never actually be run, given the redundancy, chance for error, and the ability to construct a data model in XML or other more generalizable file formats. However, once the data are ingested into a SAS data set, they can be written to an XML file with the following code:

```
* change this location to the location of the XML file to be created;
libname xmlout xml '/folders/myfolders/DSM-5_data_model.xml';

data DSMmodel;
   infile datalines delimiter=',';
   length class1 $50 class2 $50 dsm5name $100 dsm5code $8;
   input class1 $ class2 $ dsm5name $ dsm5code $;
   datalines;
Substance use and addictive disorders,Alcohol-related disorders,Alcohol dependence
with intoxication delirium,291.0
Substance use and addictive disorders,Alcohol-related disorders,Alcohol dependence
with alcohol-induced sleep disorder,291.81
Substance use and addictive disorders,Alcohol-related disorders,Unspecified alcohol-
related disorder,291.89
Substance use and addictive disorders,Substance-related disorders,Substance
dependence with intoxication delirium,292.81
Substance use and addictive disorders,Substance-related disorders,Stimulant use
disorder,304.40
Substance use and addictive disorders,Substance-related disorders,Other or unknown
substance-related disorder,304.90
;

data xmlout.DSMmodel;
    set DSMmodel;
run;
```

Note that the LIBNAME statement must be modified to a logical folder accessible by the user. The XML file (DSM-5_data_model.xml) that is produced by SAS follows:

```
  <?xml version="1.0" encoding="utf-8" ?>
- <TABLE>
- <DSMMODEL>
      <class1>Substance use and addictive disorders</class1>
      <class2>Alcohol-related disorders</class2>
      <dsm5name>Alcohol dependence with intoxication delirium</dsm5name>
      <dsm5code>291.0</dsm5code>
  </DSMMODEL>
- <DSMMODEL>
      <class1>Substance use and addictive disorders</class1>
      <class2>Alcohol-related disorders</class2>
      <dsm5name>Alcohol dependence with alcohol-induced sleep disorder</dsm5name>
      <dsm5code>291.81</dsm5code>
  </DSMMODEL>
- <DSMMODEL>
      <class1>Substance use and addictive disorders</class1>
      <class2>Alcohol-related disorders</class2>
      <dsm5name>Unspecified alcohol-related disorder</dsm5name>
      <dsm5code>291.89</dsm5code>
  </DSMMODEL>
- <DSMMODEL>
      <class1>Substance use and addictive disorders</class1>
      <class2>Substance-related disorders</class2>
      <dsm5name>Substance dependence with intoxication delirium</dsm5name>
      <dsm5code>292.81</dsm5code>
  </DSMMODEL>
- <DSMMODEL>
      <class1>Substance use and addictive disorders</class1>
      <class2>Substance-related disorders</class2>
      <dsm5name>Stimulant use disorder</dsm5name>
      <dsm5code>304.40</dsm5code>
  </DSMMODEL>
- <DSMMODEL>
      <class1>Substance use and addictive disorders</class1>
      <class2>Substance-related disorders</class2>
      <dsm5name>Other or unknown substance-related disorder</dsm5name>
      <dsm5code>304.90</dsm5code>
  </DSMMODEL>
  </TABLE>
```

Unfortunately, this XML file is rectangular rather than hierarchical, and thus grossly redundant. Thus, despite all listed disorders being a subset of “Substance use and addictive disorders,” the class is redundantly displayed for each disorder rather than listed only once as a parent element within the hierarchy. Note also that the variable types and lengths have been removed, which might especially complicate the DSM-5 code that *prima facie* may appear to be numeric, but which must be stored and referenced as a character value to preserve trailing zeros.

To preserve the hierarchical structure as well as the variable attributes, SAS practitioners can create an XML map file, which defines the structure and metadata for a data set. The overall table hierarchy is specified in the <TABLE-PATH> element as “TABLE/CLASS1/CLASS2/DIAG”. A <COLUMN> element also must exist for each variable being created, with the respective variable name, hierarchy, format, and length specified in successive <COLUMN> elements. The RETAIN=”YES” attribute causes the two classification variables to persist (even when not listed), thus reducing the redundancy observed in the previous XML file in which CLASS1 and CLASS2 were specified for each observation.

The following XML map (DSM5model.map) correctly shows that CLASS2 is subordinate to CLASS1, disorder name (DSM5name) subordinate to CLASS2, and disorder code (DSM5code) also subordinate to CLASS2:

```
<?xml version="1.0" ?>
```

```
<SXLEMAP version="2.1">
  <TABLE name="DSM5model">
        <TABLE-PATH syntax="XPath">
          /TABLE/CLASS1/CLASS2/DIAG
         </TABLE-PATH>

        <COLUMN name="CLASS1" retain="YES">
          <PATH>/TABLE/CLASS1 </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>50</LENGTH>
         </COLUMN>

        <COLUMN name="CLASS2" retain="YES">
          <PATH>/TABLE/CLASS1/CLASS2 </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>50</LENGTH>
         </COLUMN>

        <COLUMN name="DSM5name">
          <PATH>/TABLE/CLASS1/CLASS2/DIAG/@DSM5name </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>100</LENGTH>
         </COLUMN>

        <COLUMN name="DSM5code">
          <PATH>/TABLE/CLASS1/CLASS2/DIAG/@DSM5code </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>8</LENGTH>
         </COLUMN>

  </TABLE>
</SXLEMAP>
```

Now that the data structure has been appropriately represented in the XML map file (DSM5model.map), the data from Table 3 can be represented through an XML file (DSM5model.xml) that now avoids the previous redundancy:

```
<?xml version="1.0" encoding="utf-8" ?>
<TABLE>
      <CLASS1> Substance use and addictive disorders
           <CLASS2> Alcohol-related disorders
                 <DIAG  DSM5name="Alcohol  dependence  with  intoxication  delirium"
DSM5code="291.0"/>
                 <DIAG  DSM5name="Alcohol  dependence  with  alcohol-induced  sleep
disorder" DSM5code="291.81"/>
                 <DIAG     DSM5name="Unspecified     alcohol-related     disorder"
DSM5code="291.89"/>
           </CLASS2>
           <CLASS2> Substance-related disorders
                 <DIAG  DSM5name="Substance  dependence  with  intoxication  delirium"
DSM5code="292.81"/>
                 <DIAG DSM5name="Stimulant use disorder" DSM5code="304.40"/>
                 <DIAG  DSM5name="Other  or  unknown  substance-related  disorder"
DSM5code="304.90"/>
           </CLASS2>
    </CLASS1>
</TABLE>
```

With the XML map file (DSM5model.map) and XML file (DSM5model.xml) both saved in /folders/myfolders, the following code ingests the XML model according to the XML map and generates the DSMmodel data set:

```
* change this location to the location of the XML map file and XML model file;
filename DSM5in '/folders/myfolders/DSM5model.xml';
filename DSMmap '/folders/myfolders/DSM5model.map';

libname DSM5in xmlv2 xmlmap=DSMmap;

data DSMmodel;
    set DSM5in.DSM5model;
run;
```

Note that the DSMmodel data set created via the XML import is identical to the DSMmodel data set created previously using the DATALINES statement. While the XML method requires the additional definition of the XML map to specify the hierarchy and variable formats, modularity is improved because the data model is no longer defined inside the software but rather within XML. Moreover, the XML files can be used not only by SAS but also by other applications.

CREATING HIERARCHICAL SAS FORMATS (THE LAY WAY)

Table 3 represents an abridged version of the DSM-5 while highlighting the hierarchical nature of the data. In the previous section, this data model was abstracted into the DSMmodel data set using alternative methods—hardcoding the data model with the DATALINES statement, and the more dynamic (and preferred) method that ingests an XML model. A data model could also be imported from an Excel spreadsheet or a raw text file, but regardless of which method is used, the DSMmodel data set is the substrate from which hierarchical formats can be generated with ease.

To illustrate the importance of hierarchical formats, Figure 1 illustrates seven permissible use cases for transformation or binning, given the structure of the DSM-5 data model and the disorders presented in Table 3:

1. Conversion of the DSM-5 code to the DSM-5 disorder name (e.g., converting “290.0” into “Alcohol dependence with intoxication delirium”).
2. Categorization of the DSM-5 code into the DSM-5 level 2 classification (CLASS2) (e.g., categorizing “290.0” as “Alcohol-related disorders”).
3. Categorization of the DSM-5 code into the DSM-5 level 1 classification (CLASS1) (e.g., categorizing “290.0” as “Substance use and addictive disorders”).
4. Conversion of the DSM-5 disorder name to the DSM-5 code (e.g., converting “Alcohol dependence with intoxication delirium” into “290.0”).
5. Categorization of the DSM-5 disorder name into the DSM-5 level 2 classification (CLASS2) (e.g., categorizing “Alcohol dependence with intoxication delirium” as “Alcohol-related disorders”).
6. Categorization of the DSM-5 disorder name into the DSM-5 level 1 classification (CLASS1) (e.g., categorizing “Alcohol dependence with intoxication delirium” as “Substance use and addictive disorders”).
7. Categorization of the DSM-5 level 2 classification (CLASS2) into the DSM-5 level 1 classification (CLASS1) (e.g., categorizing “Alcohol-related disorders” into “Substance use and addictive disorders”).

Note that the direction of the arrow indicates a variable format being transformed or categorized into another variable format having equal or less specificity.

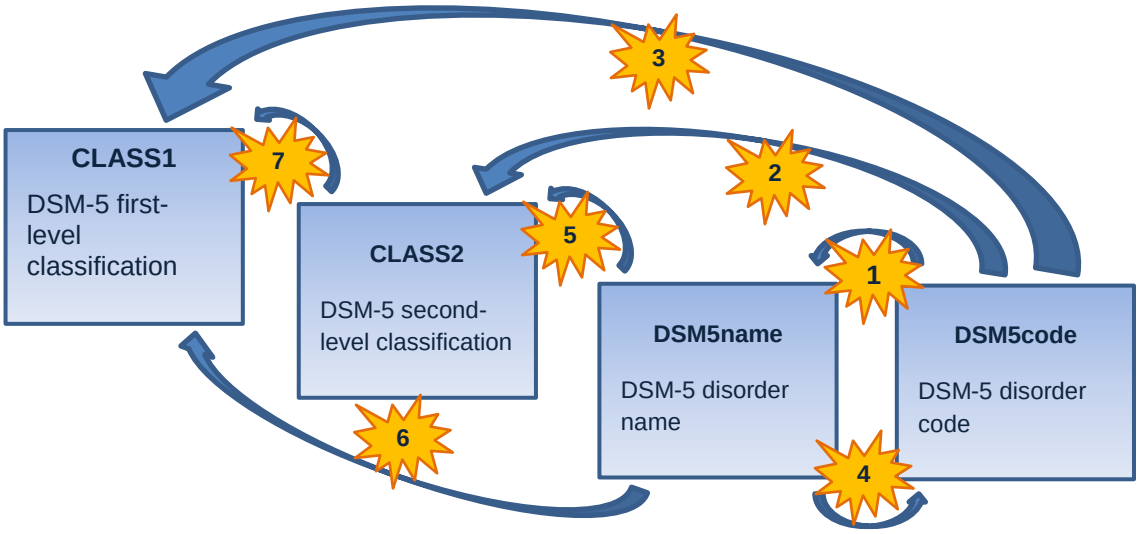

Figure 1. Permissible Transformations and Categorizations Between DSM-5 Data Model Variables

The first use case can be demonstrated in three steps, in which 1) a data set (DSM5code_only) is created that contains only a list of DSM-5 codes, 2) a format (CODE_TO_NAME) is created that transforms the DSM-5 codes into DSM-5 disorder names, and 3) a report is generated that displays both the unformatted and formatted values of the DSM5code variable:

```
data DSM5code_only;
    length DSM5code $8;
    DSM5code='291.0'; output;
    DSM5code='291.89'; output;
    DSM5code='292.81'; output;
    DSM5code='292.81'; output;
run;

proc format;
    value $ code_to_name
    '291.0'='Alcohol dependence with intoxication delirium'
    '291.82'='Alcohol dependence with alcohol-induced sleep disorder'
    '291.89'='Unspecified alcohol-related disorder'
    '292.81'='Substance dependence with intoxication delirium'
    '304.40'='Stimulant use disorder'
    '304.90'='Other or unknown substance-related disorder';
run;

proc report data=DSM5code_only;
    column DSM5code=unform DSM5code=form;
    define unform / display 'Unformatted';
    define form / display 'Formatted' format=$code_to_name.;
run;
```

The output from the REPORT procedure shows two views of the single variable DSM5code, both the unformatted (i.e., original) and formatted (i.e., transformed) values of the DSM-5 codes from the DSM5code_only data set:

Unformatted	Formatted
291.0	Alcohol dependence with intoxication delirium

291.89	Unspecified alcohol-related disorder
292.81	Substance dependence with intoxication delirium
292.81	Substance dependence with intoxication delirium

At first glance, the second use case appears straightforward, given that it relies on the same underlying data model as the first use case. However, while the first use case represents a one-to-one relationship between the DSM-5 code and the disorder name, the second use case represents a many-to-one relationship between the DSM-5 code and second-level disorder classification. Thus, were the FORMAT procedure alone to be relied upon for this new mapping, the CODE_TO_NAME format could not be reused and the additional CODE_TO_CLASS2_ format would need to be created—and redundantly maintained and modified when necessary:

```
proc format;
    value $ code_to_class2_
    '291.0','291.82','291.89'='Alcohol-related disorders'
    '292.81','304.40','304.90'='Substance-related disorders';
run;

proc report data=DSM5code_only;
    column DSM5code=unform DSM5code=form;
    define unform / display 'Unformatted';
    define form / display 'Formatted' format=$code_to_class2_.;
run;
```

Note that the CODE_TO_CLASS2_ format must end in an underscore (_) because SAS formats cannot end in a number. The output from the REPORT procedure shows the unformatted and formatted values of the DSM-5 codes in the DSM5code data set, now applying the CODE_TO_CLASS2_ format instead of the CODE_TO_NAME format:

Unformatted	**Formatted**
291.0	Alcohol-related disorders
291.89	Alcohol-related disorders
292.81	Substance-related disorders
292.81	Substance-related disorders

The necessity to create a new SAS format to map a different relationship within the same data model (i.e., from DSM-5 code to second-level disorder classification, rather than from DSM-5 code to disorder name) may not seem like much additional work—until you consider the hundreds of disorders included in the actual DSM-5 and the dozens of first- and second-level classifications required to categorize those disorders. In fact, while the first and second use cases are extremely similar in nature and rely exclusively on the same underlying data model, maintaining the unabridged versions of these two SAS formats would be nightmarish, not to mention the additional five use cases demonstrated in Figure 1, each of which would require yet another SAS format to be created. Moreover, while this simplistic representation of the DSM-5 supports seven use cases for transformation or binning, more complex data models could accommodate exponentially more pathways between variable formats, and thus exponentially more headaches.

CREATING HIERARCHICAL SAS FORMATS (THE SMART WAY)

But the solution lies in the complexity of the data model itself—the rigor enforced by the hierarchical structure through the XML map file. This structure ensures that the DSM-5 code and the disorder name represent a one-to-one relationship, and thus can be bidirectionally transformed into each other (i.e., use cases one and four) while all other pathways (i.e., use cases two, three, five, six, and seven) represent many-to-one relationships, and thus are unidirectional. In other words, the DSM-5 code and the disorder name contain identical specificity, with no granularity lost as one value is transformed into the other, and vice versa. However, as the DSM-5 code or the disorder name is categorized into either the CLASS2 or

CLASS1 variable formats, additional specificity is lost at each jump. For example, if the DSM-5 code is known, you will always know the CLASS2 and CLASS1 values; however, if only the CLASS2 value is known, you will always know the CLASS1 value yet never know either the DSM-5 code or the disorder name, because those values have greater specificity than CLASS2.

In the "Modularity" section, the DSMmodel data set was created from an XML file (and map file) to represent an abridged version of the DSM-5 data model. The previous section demonstrates seven use cases (data transformations and categorizations) that can be achieved through SAS formats given this data model. This section demonstrates how to create these seven formats automatically using the CNTLIN option of the FORMAT procedure, thus obviating the painful process of manually creating and maintaining multiple FORMAT procedures. This dynamism also greatly increases the reliability of formats through MDM principles, as a single data model is used to generate all format transformations, and thus only one model must be maintained.

The BUILD_FORMAT macro (included in Appendix A) dynamically creates a SAS format that transforms (or bins) one variable (within a defined data model) into a second variable format (within the same data model). The macro flexibly accommodates various hierarchical structures and data models of various content, so while applied here to create formats that represent the DSM-5 domain and taxonomy, it is equally applicable to other industries, organizations, knowledge domains, and use cases. While the macro validates neither the structure nor content of the XML map file (or associated XML model files), minimal exception handling is defined and represented later in this text.

The BUILD_FORMAT macro definition follows:

```
* this macro accepts either flat or hierarchical data models;
%macro build_format(fmtname= /* name of SAS format generated */,
    dsnmodel= /* data set in LIB.DSN or DSN format containing data model */,
    var1= /* variable (within the model) being transformed or categorized */,
    var2= /* variable (within the model) to which VAR1 is transformed */);
```

The BUILD_FORMAT macro parameters include:

- **FMTNAME** – This parameter is alphanumeric and must conform to SAS format-naming conventions (e.g., cannot end in a number, must be less than 32 characters). The FMTNAME represents the name of the format being created (by transforming the VAR1 variable format into the VAR2 variable format). For example, to improve readability, DSM5CODE_TO_NAME is used to represent the format that transforms a DSM-5 code into a DSM-5 disorder name.
- **DSNMODEL** – This parameter represents the SAS data set name of the data model. Both LIB.DSN format (e.g., SOMELIB.Somedata) and DSN format (e.g., Somedata, which implies WORK.Somedata) are valid. The order of columns is unimportant within the data model, but for simplicity, all data models within this text can be read from left to right, with specificity increasing (or remaining constant) when moving to the right.
- **VAR1** – This parameter is the unformatted variable that is being transformed or categorized into the formatted variable format (i.e., VAR2 parameter) via a one-to-one or many-to-one relationship as defined within the data model.
- **VAR2** – This parameter is the formatted variable format into which an unformatted variable (i.e., VAR1 parameter) is being transformed or categorized.

For example, the following sample macro invocation dynamically creates the CODE_TO_NAME format that was generated manually in the previous section (for the first use case in Figure 1), renaming it DSM5CODE_TO_NAME for comparison:

```
%build_format(fmtname=DSM5code_to_name,
    dsnmodel=DSMmodel,
    var1=DSM5code,
    var2=DSM5name);

proc report data=DSM5code_only;
    column DSM5code=unform DSM5code=form;
    define unform / display 'Unformatted';
    define form / display 'Formatted' format=$DSM5code_to_name.;
run;
```

The REPORT procedure generates output identical to that produced previously with the CODE_TO_NAME format:

Unformatted	Formatted
291.0	Alcohol dependence with intoxication delirium
291.89	Unspecified alcohol-related disorder
292.81	Substance dependence with intoxication delirium
292.81	Substance dependence with intoxication delirium

The format was derived solely from the DSM-5 data model (DSMmodel data set, produced from the XML file and XML map file) and the DSM5code data set (that represents data that might be found in clinical records which need to be transformed). Thus, the grueling step of format creation is not only automated but also made more reliable and reusable. And, should the data model need to be changed, modifications can be made at the source (within the XML map or XML file or both), driving MDM objectives and interoperability of the data model to support various uses and software applications.

To demonstrate another example, the following macro invocation creates the DSM5CODE_TO_CLASS2_ format, created manually in the previous section as the CODE_TO_CLASS2_ format:

```
%build_format(fmtname=DSM5code_to_class2_,
    dsnmodel=DSMmodel,
    var1=DSM5code,
    var2=CLASS2);

proc report data=DSM5code_only;
    column DSM5code=unform DSM5code=form;
    define unform / display 'Unformatted';
    define form / display 'Formatted' format=$DSM5code_to_class2_.;
run;
```

The REPORT procedure generates output identical to that produced previously with the CODE_TO_CLASS2 format:

Unformatted	Formatted
291.0	Alcohol-related disorders
291.89	Alcohol-related disorders
292.81	Substance-related disorders
292.81	Substance-related disorders

OVERCOMING SAS FORMAT LIMITATIONS

The first limitation of the FORMAT procedure that this text and solution aimed to overcome was the inability of formats to be applied in reverse. For example, the CODE_TO_NAME format was manually created and the DSM5CODE_TO_NAME format was dynamically created, but what effort is required to create a reverse lookup that transforms the DSM-5 disorder name into the respective DSM-5 code? To setup this example, the following DATA step creates a data set that contains only the variable DSM5name:

```
data DSM5name_only;
    length DSM5name $100;
    DSM5name='Alcohol dependence with intoxication delirium'; output;
    DSM5name='Alcohol dependence with alcohol-induced sleep disorder'; output;
```

```
    DSM5name='Unspecified alcohol-related disorder'; output;
    DSM5name='Substance dependence with intoxication delirium'; output;
    DSM5name='Substance dependence with intoxication delirium'; output;
    DSM5name='Substance dependence with intoxication delirium'; output;
run;
```

The BUILD_FORMAT macro now creates a format (DSM5NAME_TO_CODE) that can be used to transform disorder names into DSM-5 codes, should this ever be required:

```
%build_format(fmtname=DSM5name_to_code,
    dsnmodel=DSMmodel,
    var1=DSM5name,
    var2=DSM5code);

proc report data=DSM5name_only;
    column DSM5name=unform DSM5name=form;
    define unform / display 'Unformatted';
    define form / display 'Formatted' format=$DSM5name_to_code.;
run;
```

The output from the REPORT procedure demonstrates this reverse lookup applied to the DSM5name_only data set:

Unformatted	Formatted
Alcohol dependence with intoxication delirium	291.0
Alcohol dependence with alcohol-induced sleep disorder	291.81
Unspecified alcohol-related disorder	291.89
Substance dependence with intoxication delirium	292.81
Substance dependence with intoxication delirium	292.81
Substance dependence with intoxication delirium	292.81

This same methodology also overcomes the second FORMAT limitation—the inability for a one-to-one-to-one relationship (e.g., disorder name to DSM-5 code to ICD-10 code) to be represented. For example, the ICD-10 codes could be added to this model with only a few steps yet without needing to modify the BUILD_FORMAT macro:

1. Modify the XML map to include the additional COLUMN definition for ICD-10 codes:

```
<COLUMN name="ICD10code">
  <PATH>/TABLE/CLASS1/CLASS2/DIAG/@ICD10code </PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>8</LENGTH>
 </COLUMN>
```

2. Modify any XML files that rely on the XML map file to include values for ICD-10 codes. Only the DIAG element would require update, so the remainder of the structure and content can remain unchanged. For example, the following original DIAG element:

```
<DIAG DSM5name="Alcohol dependence with intoxication delirium"
   DSM5code="291.0"/>
```

   becomes:

```
<DIAG DSM5name="Alcohol dependence with intoxication delirium"
   DSM5code="291.0" ICD10code="F10.221"/>
```

   and so on until each DIAG element has an ICD10code attribute, after which BUILD_FORMAT can be run.

While not demonstrated, with ICD-10 codes now added to the data model, BUILD_FORMAT could be used to create SAS formats that could standardize either DSM-5 or ICD-10 codes into disorder names, and which could classify them further in higher-level disorder groups.

The third limitation was the inability of manual format creation and maintenance to scale efficiently as the size and complexity of data models increase. However, as previously demonstrated, the BUILD_FORMAT macro can dynamically create a format that represents any permissible relationship between two variables in the underlying data model. To demonstrate the automation and substantial reduction in effort, the following code dynamically builds formats for each of the seven use cases depicted in Figure 1:

```
%build_format(fmtname=DSM5code_to_name,
    dsnmodel=DSMmodel, var1=DSM5code, var2=DSM5name);

%build_format(fmtname=DSM5code_to_class2_,
    dsnmodel=DSMmodel, var1=DSM5code, var2=class2);

%build_format(fmtname=DSM5code_to_class1_,
    dsnmodel=DSMmodel, var1=DSM5code, var2=class1);

%build_format(fmtname=DSM5name_to_code_,
    dsnmodel=DSMmodel, var1=DSM5name, var2=DSM5code);

%build_format(fmtname=DSM5name_to_class2_,
    dsnmodel=DSMmodel, var1=DSM5name, var2=class2);

%build_format(fmtname=DSM5name_to_class1_,
    dsnmodel=DSMmodel, var1=DSM5name, var2=class1);

%build_format(fmtname=DSM5class2_to_class1_,
    dsnmodel=DSMmodel, var1=class2, var2=class1);
```

The fourth FORMAT limitation expressed was the difficulty of extracting business rules, data models, and other logic from software when they represent a knowledge domain that can or should be applied elsewhere. For example, the DSM-5 is not some esoteric data model beneficial to a single SAS program but rather an established taxonomy with endless potential uses. By operationalizing the data model within an XML file, rather than within a SAS data set or a series of FORMAT procedures, other developers or analysts who never interact with SAS can benefit from this external data model. For example, if a company represented the DSM-5 data model within XML, it could use this model to drive not only SAS analytic processes but also dynamic JavaScript menu items on its corporate website. And, were the DSM-5 to be updated (which does occur incrementally), both the SAS processes and corporate website could be updated automatically and remain synchronized.

Yet another benefit of MDM is the ability to validate data models in only one location. The BUILD_FORMAT macro performs cursory validation by ensuring that the variable-to-variable transformation that is selected (i.e., specified in the VAR1 and VAR2 macro parameters) will not violate the data model. Thus, while it's possible to transform data from one variable to a variable format of *equal or lesser* specificity, it is not possible to transform data from one variable to a variable format having *greater* specificity. For example, in the DSM-5 data model, a variable formatted as the DSM-5 disorder name can be transformed to the CLASS1 format, but a CLASS2 variable cannot be transformed to either the DSM-5 disorder name or DSM-5 code—because those formats each have greater specificity than the CLASS2 variable format.

The BUILD_FORMAT macro guards against this type of violation (i.e., attempting to represent a one-to-many relationship rather than a many-to-one relationship). For example, the following code will cause an exception and failure and the BUILD_FORMATRC (i.e., BUILD_FORMAT return code) global macro variable to be changed to "FAILURE ON VARIABLE ORDER OR XML DATA":

```
%build_format(fmtname=DSM5class2_to_code,
    dsnmodel=DSMmodel,
    var1=CLASS2,
    var2=DSM5code);
%put &build_formatRC;
```

Thus, by ensuring that data are held principally in data models rather than brittle, hardcoded solutions, data models can be validated once only at their source.

NUMERIC FORMATS

While previous examples have relied upon the DSM5model that includes only character data, the BUILD_FORMAT macro performs equivalently when SAS formats are applied to numeric data. To illustrate this, the DSM-5 disorder code can be transformed from character to numeric data. While this example is unrealistic (because DSM-5 codes can contain zeros after decimals and thus should be encoded as character data), it demonstrates the functionality of BUILD_FORMAT with numeric data. Moreover, the benefits of maintaining a centralized data model are shown in which changes can be made that will propagate throughout all downstream uses.

In this example, the XML map file (DSM5model.map) should be altered, changing the DSM5code TYPE element from character to numeric, and the DATATYPE element from string to decimal, and saving the updated file as DSM5model_numeric.map:

```
<COLUMN name="DSM5code">
  <PATH>/TABLE/CLASS1/CLASS2/DIAG/@DSM5code </PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>decimal</DATATYPE>
    <LENGTH>8</LENGTH>
 </COLUMN>
```

The XML file (DSM5model.xml) containing the data model requires no updates, and can be ingested into SAS with the following code:

```
filename DSM5in '/folders/myfolders/datadriven/DSM5model.xml';
filename DSMmap '/folders/myfolders/datadriven/DSM5model_numeric.map';

libname DSM5in xmlv2 xmlmap=DSMmap;

data DSMmodel_numeric;
    set DSM5in.DSM5model;
run;
```

A data set with the DSM5code encoded as numeric (not character) data is required, so the following code creates the DSM5code_only_numeric variable:

```
data DSM5code_only_numeric;
    length DSM5code 8;
    DSM5code=291; output;
    DSM5code=291.89; output;
    DSM5code=292.81; output;
    DSM5code=292.81; output;
run;
```

The BUILD_FORMAT macro next generates the DSM5codeNUM_to_name format, a format that reads numeric data and transforms them into a character format, after which the REPORT procedure demonstrates the unformatted numeric variable and the formatted character variable equivalent:

```
%build_format(fmtname=DSM5codeNUM_to_name,
    dsnmodel=DSMmodel_numeric,
    var1=DSM5code,
    var2=DSM5name);

proc report data=DSM5code_only_numeric;
    column DSM5code=unform DSM5code=form;
    define unform / display 'Unformatted';
    define form / display 'Formatted' format=DSM5codeNUM_to_name.;
run;
```

Note that a numeric format (as referenced by the REPORT procedure) should not be preceded by a dollar sign ($) in the FORMAT statement. The REPORT procedure produces the following output:

Unformatted	Formatted
291	Alcohol dependence with intoxication delirium
291.89	Unspecified alcohol-related disorder
292.81	Substance dependence with intoxication delirium
292.81	Substance dependence with intoxication delirium

The output is nearly identical to that produced previously; however, because trailing zeros were eliminated (both in the XML file containing the data model and the DSM5code_only_numeric data set), the output contains no trailing zeros and 291.0 is erroneously represented as 291. This doesn't represent a limitation of BUILD_FORMAT, but rather demonstrates that DSM-5 codes should be encoded as character data to maintain the necessary trailing zeros.

To perform the reverse transformation (from a character variable into a numeric format), the following format (DSM5name_to_codeNUM) can be created with BUILD_FORMAT:

```
%build_format(fmtname=DSM5name_to_codeNUM,
    dsnmodel=DSMmodel_numeric,
    var1=DSM5name,
    var2=DSM5code);

proc report data=DSM5name_only;
    column DSM5name=unform DSM5name=form;
    define unform / display 'Unformatted';
    define form / display 'Formatted' format=$DSM5name_to_codeNUM.;
run;
```

The REPORT procedure produces the following output:

Unformatted	Formatted
Alcohol dependence with intoxication delirium	291
Alcohol dependence with alcohol-induced sleep disorder	291.81
Unspecified alcohol-related disorder	291.89
Substance dependence with intoxication delirium	292.81
Substance dependence with intoxication delirium	292.81
Substance dependence with intoxication delirium	292.81

The results match the previous application of the DSM5name_to_code format, with the exception that trailing zeros are again omitted from the DSM-5 code. While not demonstrated, the transformation of a numeric variable into a new variable also having a numeric format can be similarly performed with the BUILD_FORMAT macro.

BUILD_FORMAT EXCEPTION HANDLING

To facilitate implementation into production software systems, the BUILD_FORMAT macro contains basic exception handling that returns an error code when a warning or runtime error is detected via the &SYSCC automatic macro variable. The BUILD_FORMATRC (BUILD_FORMAT return code) global macro variable is set to one of five possible values during macro execution, designed to aid in failure identification, exception handling, and error resolution:

1. **[BLANK]** – No warning or runtime error occurred during execution (i.e., &SYSCC=0) and the variable transformation requested does not appear to violate the referenced data model. For example, if the length of BUILD_FORMATRC is zero (e.g., %if %length(&build_formatRC)=0 …), successful execution is indicated.
2. **FAILURE** – This general error code is initialized at the beginning of the macro; thus, if the macro terminates abnormally for any reason before another error code can be assigned, this default will persist.
3. **FAILURE BEFORE FORMAT** – A warning or error occurred before the FORMAT procedure was attempted, which would likely be due to invalid macro parameters having been supplied.
4. **FAILURE ON VARIABLE ORDER OR XML DATA** – The most likely culprit is the attempt to perform a transformation that violates the data model (e.g., attempting to transform a variable into a variable format that has *greater* specificity than the original variable). This can occur due to a faulty XML map file or due to faulty data in the XML file or both.
5. **FAILURE DURING FORMAT** – The format statement itself failed, reflecting that the CNTLIN input data set (Format_temp1) has malformed data, or that the format could not otherwise be created.

By implementing even simple exception handling, SAS practitioners can improve the discovery of warnings and runtime errors and thus improve the reliability and robustness of BUILD_FORMAT within larger SAS software applications.

CONCLUSION

SAS formatting is a powerful tool that can be implemented with relative ease to clean, transform, or categorize data. The FORMAT procedure essentially represents a simple data model that instructs SAS how to manipulate or represent data. However, when successive transformations are required (e.g., cleaning followed by transforming), or as data models grow increasingly complex and represent hierarchical structures, the FORMAT procedure fails to represent these more complex models. Coupled with this adversity, SAS formats are also typically hardcoded in software and thus have limited maintainability, reusability, and interoperability. The BUILD_FORMAT overcomes these limitations by relying upon an external data model to create dynamic SAS formats automatically. It represents a flexible, out-of-the-box solution that can be applied across various industries, organizations, and knowledge domains.

REFERENCES

[i] Base SAS® 9.4 Procedures Guide, Seventh Edition. *PROC FORMAT Statement*. 2017. SAS Institute. Retrieved from http://documentation.sas.com/?docsetId=proc&docsetTarget=n1c16dxnndwfzyn14o1kb8a4312m.htm&docsetVersion=9.4&locale=en.

[ii] American Psychiatric Association. *Supplement to Diagnostic and Statistical Manual of Mental Disorders, Fifth Edition.* October 2017. Retrieved from https://psychiatryonline.org/pb-assets/dsm/update/DSM5Update_October2017.pdf.

[iii] ISO/IEC 25010:2011. *Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models*. Geneva, Switzerland: International Organization for Standardization and Institute of Electrical and Electronics Engineers.

[iv] Troy Martin Hughes. 2013. Binning Bombs When You're Not a Bomb Maker: A Code-Free Methodology to Standardize, Categorize, and Denormalize Categorical Data Through Taxonomical Control Tables. *Southeast SAS Users Group (SESUG).* Retrieved from http://analytics.ncsu.edu/sesug/2013/BB-04.pdf.

[v] Troy Martin Hughes. 2016. *SAS® Data Analytic Development: Dimensions of Software Quality*. John Wiley and Sons, Inc. Hoboken, NJ.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

APPENDIX A. BUILD_FORMAT MACRO

```
%macro build_format(fmtname= /* name of SAS format generated */,
    dsnmodel= /* data set in LIB.DSN or DSN format containing data model */,
    var1= /* variable (within the model) being transformed or categorized */,
    var2= /* variable (within the model) to which VAR1 is transformed */);
%global build_formatRC;
%let build_formatRC=FAILURE;
%let syscc=0;
%local lib dsn len1 type1 len2 type1 type2 obs1 obs2;
* parse LIB and DSN if both provided, otherwise assign WORK to library;
%if %length(%scan(&dsnmodel,2,.))=0 %then %do;
    %let lib=WORK;
    %let dsn=%upcase(&dsnmodel);
    %end;
%else %do;
    %let lib=%upcase(%scan(&dsnmodel,1,.));
    %let dsn=%upcase(%scan(&dsnmodel,2,.));
    %end;
* determine the formats and lengths of VAR1 and VAR2;
proc sql noprint;
    select length, lowcase(substr(type,1,1)) into :len1, :type1
           from dictionary.columns
           where upcase(libname)="&lib" and upcase(memname)="&dsn" and
                 upcase(name)="%upcase(&var1)";
    select length, lowcase(substr(type,1,1)) into :len2, :type2
           from dictionary.columns
           where upcase(libname)="&lib" and upcase(memname)="&dsn" and
                 upcase(name)="%upcase(&var2)";
    quit;
* delete temp data sets if they already exist;
%if %sysfunc(exist(format_temp1)) %then %do;
    proc delete data=format_temp1;
    %end;
%if %sysfunc(exist(format_temp2)) %then %do;
    proc delete data=format_temp2;
    %end;
* extract values from model for dynamic format;
data format_temp1;
    length fmtname $32 type $2;
    length start %sysfunc(ifc(%substr(&type1,1,1)=c,$,)) &len1 label
           %sysfunc(ifc(%substr(&type2,1,1)=c,$,)) &len2;
    set &dsnmodel (keep=&var1 &var2 rename=(&var1=start &var2=label));
    fmtname="&fmtname";
    type="&type1";
run;
* remove duplicate values if they were created during binning;
proc sort data=format_temp1 nodupkey;
    by start label;
run;
* fail if LABEL variable is duplicate (invalid SQL or invalid format);
proc sort data=format_temp1 out=format_temp2 nodupkey;
    by start;
run;
proc sql noprint;
```

```
    select count(*)
           into :obs1
           from format_temp1;
    select count(*)
           into :obs2
           from format_temp2;
    quit;
%if &syscc=0 %then %do;
    %if &obs1=&obs2 %then %do;
           * create format;
           proc format cntlin=format_temp1;
           run;
           %if &syscc=0 %then %let build_formatRC=;
           %else %let build_formatRC=FAILURE DURING FORMAT;
           %end;
    %else %let build_formatRC=FAILURE ON VARIABLE ORDER OR XML DATA;
    %end;
%else %let build_formatRC=FAILURE BEFORE FORMAT;
%mend;
```