Fuzzy Matching Programming Techniques Using SAS® Software

Kirk Paul Lafler, Software Intelligence Corporation
Stephen Sloan, Accenture

Abstract

Data comes in all forms, shapes, sizes and complexities. Stored in files and data sets, SAS® users across industries know all too well that data can be, and often is, problematic and plagued with a variety of issues. When unique and reliable identifiers are available, users routinely are able to match records from two or more data sets using merge, join, and/or hash programming techniques without problem. But, what happens when a unique identifier, referred to as the key, is not reliable or does not exist. These types of problems are common and are found in files containing a subscriber name, mailing address, and/or misspelled email address, where one or more characters are transposed, or are partially and/or incorrectly recorded? This paper and presentation introduces what fuzzy matching is, a sampling of data issues users have to deal with, popular data cleaning and user-defined validation techniques, the application of the CAT functions, the SOUNDEX (for phonetic matching) algorithm, SPEDIS, COMPGED, and COMPLEV functions, and an assortment of programming techniques to resolve key identifier issues and to successfully merge, join and match less than perfect or messy data.

Introduction

When data sources and data sets contain consistent and valid data values, share common unique identifier(s), and have no missing data, the matching process rarely presents any problems. But, when data originating from multiple sources contain duplicate observations, duplicate and/or unreliable keys, missing values, invalid values, capitalization and punctuation issues, inconsistent matching variables, and imprecise text identifiers, the matching process is often compromised by unreliable and/or unpredictable results. When issues like these exist, SAS users must first clean and standardize any and all data irregularities before any attempts to match data records are performed. To assist in this time-consuming and costly process, users often utilize special-purpose programming techniques including the application of one or more SAS functions, the use of approximate string matching, and/or an assortment of constructive programming techniques to standardize and combine data sets together.

Data Sets Used in Examples

The examples presented in this paper illustrate two data sets, Movies_with_Messy_Data and Actors_with_Messy_Data. The Movies_with_Messy_Data data set, illustrated in Figure 1, consists of 31 observations, a data structure of six variables where Title, Category, Studio, and Rating are defined as character variables; and Length and Year are defined as numeric variables. After careful inspection several data issues can be found in this data set including the existence of missing data, duplicate observations, spelling errors, punctuation inconsistencies, and invalid values.

The Actors_with_Messy_Data data set, illustrated in Figure 2, contains 15 observations and a data structure consisting of three character variables: Title, Actor_Leading and Actor_Supporting. As with the Movies_with_Messy_Data data set, several data issues are found including missing data, spelling errors, punctuation inconsistencies, and invalid values.

	Title	Length	Category	Year	Studio	Rating
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
2	Brave Heart	177	Acton Adventure	1995	Paramont Pictures	R
3	Casablanca	103	Drama	1942	MGM / UA	PG
4	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
5	Coming to America	116	Comedy	1988	Paramount Pictures	R
6	Dracula	130	Horror	1993	Columbia TriStar	R
7	Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
8	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
9	Forrest Gump	143	Drama	1994	Paramount Pictures	PG-13
10	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
11	Jaws	125	Action Adventure	1975	Universal Studios	PG
12	Jurassic Park	127	Action	1993	Universal Pictures	PG-13
13	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
14	Michael	106	Drama	1997	Warner Bros	PG-13
15	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
16	Poltergeist	115	Horror	1982	MGM / UA	PG
17	Rocky	120	Action Adventure	1976	MGM / UA	PG
18	Scarface	170	Action Cops & Robber	1983	Universal Studios	r
19	Silence of the Lambs	118	Drama Suspense	1991	Orion	R
20	Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
21	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	GP
22	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
23	The Wizard of Oz	101	Adventure	1939	MGM / UA	G
24	The Wizard of Ozz	102	Adventure	1939	MGM - UA	g
25	Titanic	194	Dramma Romance	1997	Paramount Pictures	PG-13
26	Rocky	120	Action Adventure	1976	MGM / UA	PG
27	Forrest Gumpp	143	Dramma	1994	Paramont Pictures	PG13
28	Christmas Vacatiion	97	Commedy	1989	Warner Brothers	PG-13
29	National Lampoons Vacation	98	Comedy	1983	Warner Brothers	PG-13
30	Micheal	106	Drama	1997	Warner Brothers	PG-13
31		177	Acton Adventure	1995	Paramont Pictures	R

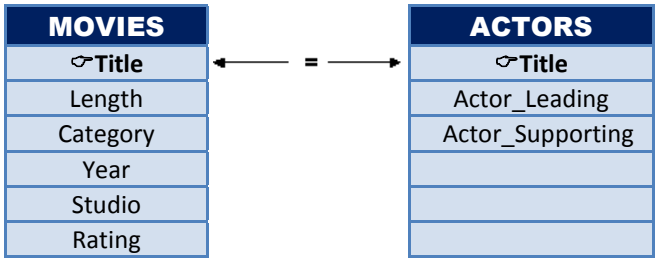
Figure 1. Movies_with_Messy_Data data set.

	Title	Actor_Leading	Actor_Supporting
1	Brave Heart	Mel Gibson	Sophie Marceau
2	XMAS Vacation	Chevy Chase	Beverly D'Angelo
3	Coming to America	Eddie Murphy	Arsenio Hall
4	Forrest Gump	Tom Hanks	Sally Field
5	GHOST	Patrick Swayze	Demi Moore
6	Lethal Weapon	Mel Gibson	Danny Glover
7	Michael	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	Chevy Chase	Beverly D'Angelo
9	Rocky	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	Anthony Hopkins	Jodie Foster
11	The Hunt for Red Oktober	Sean Connery	Alec Baldwin
12	The Terminator	Arnold Schwarzenegge	Michael Biehn
13	Titanic	Leonardo DiCaprio	Kate Winslet
14		Mell Gibson	Sophie Marceau
15	National Lampoons Vacation	Chevy Chase	Beverly D'Angelo

Figure 2. Actors_with_Messy_Data data set.

The Matching Process Explained

In an age of endless spreadsheets, apps and relational database management systems (RDBMS), it's unusual to find a single sheet, file, table or data set that contains all the data needed to answer an organization's questions. Today's data exists in many forms and all too often involves matching two or more data sources to create a combined file. The matching process typically involves combining two or more data sets, spreadsheets and/or files possessing a shared, common and reliable, identifier (or key) to create a single data set, spreadsheet and/or file. The matching process, illustrated in the following diagram, shows two tables with a key, Title, to combine the two tables together.



But, when a shared and reliable key is associated with input data sources that are nonexistent, inexact, or unreliable, the matching process often becomes more involved and problematic. As cited in Sloan and Hoicowitz (2016), special processes are needed to successfully match the names and addresses from different files when they are similar, but not exactly the same. In a constructive and systematic way the authors of this paper describe a six step approach to cleansing data and performing fuzzy matching techniques.

Step 1: Remove extraneous characters.

As a general rule, punctuation can differ while the names are the same. For example, John's "super" pizza and John's super pizza refer to the same restaurant. Therefore, we remove the following characters from all names: ' " & ? - .

Step 2: Put all characters in upper-case notation and remove leading blanks.

Step 3: Remove words that might or might not appear in the same company name.

Some examples are The, .com, Inc, LTD, LLC, DIVISION, CORP, CORPORATION, CO., and COMPANY.

Step 4: Rationalize the zip codes when matching addresses.

We found it useful to remove the last 4 digits of 9-digit zip codes, because some files might only have 5-digit zip codes. Since some files might have zip codes as numeric fields, and other files might have zip codes as character fields, make sure to include leading zeroes. For example, zip codes with a leading zero, as in 08514, would appear in a numeric field as 8514 requiring the leading zero to be inserted.

If working with US zip codes, make sure they are all numeric. This may not apply for other countries. One common mistake to watch for is that sometimes Canada, with abbreviation CA, is put in as the state CA (California) instead of the country CA. Since Canada has an alphanumeric 6-character zip code, this, hopefully, will be caught when checking for numeric zip codes.

Step 5: Choose a standard for addresses.

Decide whether to use Avenue or Ave, Road or Rd, etc, and then convert the address fields to match the standard.

Step 6: Match the names and addresses using one or more fuzzy matching techniques.

Users have an assortment of powerful SAS algorithms, functions and programming techniques to choose from.

Fuzzy matching is the process by which data is combined where a known key either does not exist and/or the variable(s) representing the key is/are unreliable. In Dunn (2014), the author suggests addressing these types of scenarios using the following steps.

1. Determine the likely matching variables using metadata (e.g., PROC CONTENTS, etc.) listings.
2. Perform data cleaning.
3. Use the COMPGED function to determine the dissimilarity between two strings.

The authors of this paper agree with Sloan & Hoicowitz, and Dunn's strategies for handling fuzzy matching issues. But, we also want to stress the importance of understanding the physical side of data along with the distribution of data values. To address these areas, we suggest adhering to a five step approach, as follows:

1. Determine the likely matching variables using metadata (e.g., PROC CONTENTS, etc.) listings.
2. Understand the distribution of data values including the number of levels for categorical and key variables.
3. Perform data cleaning.
4. Perform data transformations.
5. Use Fuzzy matching programming techniques when a reliable key between data sources are nonexistent, inexact or unreliable.

Step #1: Determining the Likely Matching Variables

This first step determines whether any variables exist for matching purposes. Using a PROC CONTENTS alphabetical list of variables and attributes listing for the data sets, Movies_with_Messy_Data and Actors_with_Messy_Data, shown below; compare each variable assessing the likelihood of potential matching variables. The PROC CONTENTS code is illustrated below.

PROC CONTENTS Code:

```
proc contents data=mydata.Movies_with_Messy_Data ;
run ;
proc contents data=mydata.Actors_with_Messy_Data ;
run ;
```

From the PROC CONTENTS listing, illustrated in Figure 3, we see that TITLE is consistently defined in both data sets as a $30 character variable. Based on this, we examine the values of the TITLE variable in greater detail to determine whether it can serve as the key for matching observations in both data sets, as well as the distribution of data values for other categorical variables.

Movies_with_Messy_Data

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
3	Category	Char	20
2	Length	Num	3
6	Rating	Char	5
5	Studio	Char	25
1	Title	Char	30
4	Year	Num	4

Actors_with_Messy_Data

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	Actor_Leading	Char	20
3	Actor_Supporting	Char	20
1	Title	Char	30

Figure 3. PROC CONTENTS Metadata Results.

Step #2: Understanding the Distribution of Data Values and NLEVELS

To derive a more accurate picture of the data sources, we suggest that users conduct extensive data analysis by identifying missing values, outliers, invalid values, minimum and maximum values, averages, value ranges, duplicate observations, distribution of values, and the number of distinct values a categorical variable contains. This important step provides an understanding of the data, while leveraging the data cleaning and standardizing activities that will be performed later. One of the first things data wranglers will want to do is explore the data using the FREQ procedure.

PROC FREQ Code:

```
proc freq data=mydata.Movies_with_Messy_Data ;
  tables Title / NOCUM NOPERCENT
           out=Missing_Titles(where=(Title = "")) ;
run ;
```

Reviewing the FREQ results, we see there are duplicate “key” values and missing values, as shown in Figure 4.

Title	Frequency
Brave Heart	2
Casablanca	1
Christmas Vacatiion	1
Christmas Vacation	1
Coming to America	1
Dracula	1
Dressed to Kill	1
Forrest Gump	2
Forrest Gumpp	1
Ghost	1
Jaws	1
Jurassic Park	1
Lethal Weapon	1
Michael	1
Micheal	1
National Lampoon's Vacation	1
National Lampoons Vacation	1
Poltergeist	1
Rocky	2
Scarface	1
Silence of the Lambs	1
Star Wars	1
The Hunt for Red October	1
The Terminator	1
The Wizard of Oz	1
The Wizard of Ozz	1
Titanic	1
Frequency Missing = 1	

Figure 4. PROC FREQ Results show duplicate “key” values and missing values.

Determining the number of distinct values a categorical variable has is critical knowledge that all data analysts and wranglers seek an answer to. Acquiring this information helps everyone involved to better understand the number of distinct variable levels, the unique values and the number of occurrences for developing data-driven programming constructs and elements. The FREQ procedure provides details about the number of levels for each categorical variable.

PROC FREQ Code:

```
title "NLevels for Variables of Interest in Movies_with_Messy_Data" ;
proc freq data=mydata.Movies_with_Messy_Data nlevels ;
  tables Title Rating Category Studio / nopct nocum ;
run ;
```

Reviewing the PROC FREQ results, we see the distinct variable levels for each variable: Title, Rating, Category and Studio, as shown in Figure 5.

NLevels for Variables of Interest in Movies_with_Messy_Data

The FREQ Procedure

Number of Variable Levels			
Variable	Levels	Missing Levels	Nonmissing Levels
Title	28	1	27
Rating	8	0	8
Category	15	0	15
Studio	13	0	13

Figure 5. PROC FREQ results show the number of levels for each variable of interest.

Reviewing the PROC FREQ results, an assortment of data consistency, validation and capitalization issues have been identified for each variable, as shown in Figure 6.

Rating	Frequency
G	1
GP	1
PG	6
PG-13	11
PG13	1
R	9
g	1
r	1

Category	Frequency
Action	1
Action Adventure	5
Action Cops & Robber	2
Action Sci-Fi	2
Acton Adventure	2
Adventure	2
Comedy	4
Commedy	1
Drama	5
Drama Mysteries	1
Drama Romance	1
Drama Suspense	1
Dramma	1
Dramma Romance	1
Horror	2

Studio	Frequency
Columbia TriStar	1
Filmways Pictures	1
Live Entertainment	1
Lucas Film Ltd	1
MGM - UA	1
MGM / UA	5
Orion	1
Paramont Pictures	3
Paramount Pictures	7
Universal Pictures	1
Universal Studios	2
Warner Bros	1
Warner Brothers	6

Figure 6. PROC FREQ results depict unique values and the number of occurrences for each variable of interest.

Step #3: Performing Data Cleaning

Data cleaning, often referred to as data scrubbing, is the process of identifying and fixing data quality issues including missing values, invalid character and numeric values, outlier values, value ranges, duplicate observations, and other anomalies found in data sets. SAS provides many powerful ways to perform data cleaning tasks. For anyone wanting a complete guide to the various SAS data cleaning techniques, we highly recommend Cody's Data Cleaning Techniques Using SAS, Third Edition. To illustrate one popular data cleaning technique that users frequently turn to for identifying and removing duplicate observations, we illustrate the SORT procedure.

Exploring PROC SORT to Identify and Remove Duplicate Observations

A popular approach with users for identifying and removing duplicate observations in a data set is to use PROC SORT. By using the SORT procedure's three options: **DUPOUT=**, **NODUPRECS**, and **NODUPKEYS**, users are better able to control how duplicate observations are identified and removed.

Specifying the DUPOUT= Option

PROC SORT's **DUPOUT=** option is often used to identify duplicate observations before actually removing them from a data set. A DUPOUT= option, often specified when a data set is too large for visual inspection, can be used with the NODUPKEYS or NODUPRECS options to name a data set that contains duplicate keys or entire observations. In the next example, the DUPOUT=, OUT= and NODUPKEY options are specified to identify duplicate keys.

PROC SORT Code:

```
PROC SORT DATA=mydata.Movies_with_Messy_Data
        DUPOUT=Movies_Dupout_NoDupkey
```

```
            OUT=Movies_Sorted_Cleaned_NoDupkey
         NODUPKEY ;
  BY Title ;
RUN ;

PROC PRINT DATA=work.Movies_Dupout_NoDupkey NOOBS ;
  TITLE "Observations Slated for Removal" ;
RUN ;
PROC PRINT DATA=work.Movies_Sorted_Cleaned_NoDupkey NOOBS ;
  TITLE "Cleaned Movies Data Set" ;
RUN ;
```

Results:

Observations Slated for Removal

Title	Length	Category	Year	Studio	Rating
Brave Heart	177	Acton Adventure	1995	Paramont Pictures	R
Forrest Gump	143	Drama	1994	Paramount Pictures	PG-13
Rocky	120	Action Adventure	1976	MGM / UA	PG

Cleaned Movies Data Set

Title	Length	Category	Year	Studio	Rating
	177	Acton Adventure	1995	Paramont Pictures	R
Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
Casablanca	103	Drama	1942	MGM / UA	PG
Christmas Vacatiion	97	Commedy	1989	Warner Brothers	PG-13
Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
Coming to America	116	Comedy	1988	Paramount Pictures	R
Dracula	130	Horror	1993	Columbia TriStar	R
Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
Forrest Gumpp	143	Dramma	1994	Paramont Pictures	PG13
Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
Jaws	125	Action Adventure	1975	Universal Studios	PG
Jurassic Park	127	Action	1993	Universal Pictures	PG-13
Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
Michael	106	Drama	1997	Warner Bros	PG-13
Micheal	106	Drama	1997	Warner Brothers	PG-13
National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
National Lampoons Vacation	98	Comedy	1983	Warner Brothers	PG-13
Poltergeist	115	Horror	1982	MGM / UA	PG
Rocky	120	Action Adventure	1976	MGM / UA	PG
Scarface	170	Action Cops & Robber	1983	Universal Studios	r
Silence of the Lambs	118	Drama Suspense	1991	Orion	R
Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	GP
The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
The Wizard of Oz	101	Adventure	1939	MGM / UA	G
The Wizard of Ozz	102	Adventure	1939	MGM - UA	g
Titanic	194	Dramma Romance	1997	Paramount Pictures	PG-13

PROC SORT's **NODUPRECS** (or **NODUPREC**) (or **NODUP**) option identifies observations with identical values for all columns. In the next example, the **OUT=, DUPOUT=** and **NODUPRECS** options are specified.

PROC SORT Code:

```
PROC SORT DATA=mydata.Movies_with_Messy_Data
        DUPOUT=Movies_Dupout_NoDupRecs
           OUT=Movies_Sorted_Cleaned_NoDuprecs
```

```
        NODUPRECS ;
  BY Title ;
RUN ;

PROC PRINT DATA=work.Movies_Dupout_NoDuprecs NOOBS ;
  TITLE "Observations Slated for Removal" ;
RUN ;

PROC PRINT DATA=work.Movies_Sorted_Cleaned_NoDuprecs NOOBS ;
  TITLE "Cleaned Movies Data Set" ;
RUN ;
```

Results:

Observations Slated for Removal

Title	Length	Category	Year	Studio	Rating
Rocky	120	Action Adventure	1976	MGM / UA	PG

Cleaned Movies Data Set

Title	Length	Category	Year	Studio	Rating
	177	Acton Adventure	1995	Paramont Pictures	R
Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
Brave Heart	177	Acton Adventure	1995	Paramont Pictures	R
Casablanca	103	Drama	1942	MGM / UA	PG
Christmas Vacatiion	97	Commedy	1989	Warner Brothers	PG-13
Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
Coming to America	116	Comedy	1988	Paramount Pictures	R
Dracula	130	Horror	1993	Columbia TriStar	R
Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
Forrest Gump	143	Drama	1994	Paramount Pictures	PG-13
Forrest Gumpp	143	Dramma	1994	Paramont Pictures	PG13
Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
Jaws	125	Action Adventure	1975	Universal Studios	PG
Jurassic Park	127	Action	1993	Universal Pictures	PG-13
Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
Michael	106	Drama	1997	Warner Bros	PG-13
Micheal	106	Drama	1997	Warner Brothers	PG-13
National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
National Lampoons Vacation	98	Comedy	1983	Warner Brothers	PG-13
Poltergeist	115	Horror	1982	MGM / UA	PG
Rocky	120	Action Adventure	1976	MGM / UA	PG
Scarface	170	Action Cops & Robber	1983	Universal Studios	r
Silence of the Lambs	118	Drama Suspense	1991	Orion	R
Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG
The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	GP
The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R
The Wizard of Oz	101	Adventure	1939	MGM / UA	G
The Wizard of Ozz	102	Adventure	1939	MGM - UA	g
Titanic	194	Dramma Romance	1997	Paramount Pictures	PG-13

Note: Although the removal of duplicates using PROC SORT is a popular technique among many SAS users, an element of care should be given to using this method when processing large data sets. Since sort operations can often be CPU-intensive operations, the authors of this paper recommend comparing PROC SORT to procedures like PROC SUMMARY with the CLASS statement to determine the performance impact of one method versus another.

Exploring SAS Functions to Modify Data

SAS functions are an essential component of the SAS Base software. Representing a variety of built-in and callable routines, functions serve as the "work horses" in the SAS software providing users with "ready-to-use" tools designed to ease the burden of writing and testing often lengthy and complex code for a variety of programming tasks. The advantage of using SAS functions is evident by their relative ease of use, and their ability to provide a more efficient, robust and scalable approach to simplifying a process or programming task.

SAS functions span a number of functional categories, including character, numeric, character string matching, data concatenation, truncation, data transformation, search, date and time, arithmetic and trigonometric, hyperbolic, state and zip code, macro, random number, statistical and probability, financial, SAS file I/O, external files, external routines, sort, to name a few. The next example illustrates an old, an alternate, and new way of concatenating strings and/or variables together. The code, results and analysis appear below.

DATA Step and CAT Functions:

```
data _null_ ;
  length NUM 3. A B C D E $ 8 BLANK $ 1 ;
  A = 'The' ;
  NUM = 5 ;
  B = ' Cats' ;
  C = 'in' ;
  D = ' the' ;
  E = 'Hat' ;
  BLANK = ' ' ;

 * Old way of concatenating with TRIM and LEFT functions and concatenation operator ;
  OLD = trim(left(A)) || BLANK || trim(left(NUM)) || BLANK || trim(left(B)) ||
        BLANK || trim(left(C)) || BLANK || trim(left(D)) || BLANK || trim(left(E)) ;

 * Using the STRIP function and concatenation operator ;
  STRIP = strip(A) || BLANK || strip(NUM) || BLANK || strip(B) || BLANK ||
          strip(C) || BLANK || strip(D)   || BLANK || strip(E) ;

 * Using the CAT functions to concatenate character and numeric values together ;

❶ CAT  = cat (A, NUM, B, C, D, E) ;
❷ CATQ = catq(BLANK, A, NUM, B, C, D, E) ;
❸ CATS = cats(A, NUM, B, C, D, E) ;
❹ CATT = catt(A, NUM, B, C, D, E) ;
❺ CATX = catx(BLANK, A, NUM, B, C, D, E) ;
   put OLD= / STRIP= / CAT= / CATQ= / CATS= / CATT= / CATX= / ;
run ;
```

Results:

```
OLD=The 5 Cats in the Hat
STRIP=The 5 Cats in the Hat
CAT=The     5 Cats   in       the    Hat
CATQ="The     " 5 " Cats   " "in      " " the    " "Hat     "
CATS=The5CatsintheHat
CATT=The5 Catsin theHat
CATX=The 5 Cats in the Hat
```

Analysis:

In the preceding SAS code, a single numeric variable, NUM, and six character variables: A, B, C, D, E, and BLANK are defined with their respective values as: NUM=5, A='The', B=' Cats', C='in', D=' the', E='Hat' and BLANK=' '. The oldest way of concatenating two or more strings or variables together is specified using the TRIM and LEFT functions and the concatenation operator "||" in an assignment statement. An alternate approach uses a STRIP function with the concatenation operator "||" in an assignment statement to join two or more strings or variables together. Finally, the newer and more robust concatenation approach uses the CAT family of functions: CAT, CATQ, CATS, CATT, and CATX.

❶ CAT, the simplest of concatenation functions, joins two or more strings and/or variables together, end-to-end producing the same results as with the concatenation (double bar) operator.

❷ CATQ is similar to the default features of the CATX function, but the CATQ function adds quotation marks to any concatenated string or variable.

❸ CATS removes all leading and trailing blanks and concatenates two or more strings and/or variables together.

❹ CATT removes trailing blanks and concatenates two or more strings and/or variables together.

❺ CATX, perhaps the most robust CAT function, removes leading and trailing blanks and concatenates two or more strings and/or variables together with a delimiter between each.

Validating Data with PROC FORMAT

Problems with data often necessitate time-consuming validation activities. The strategy is to take the time to become familiar with the data and to discover any problems before expending data analysis and reporting resources. A popular technique used by many to identify data issues is to use the FORMAT procedure. In the next example, a user-defined format is created with PROC FORMAT, a DATA step identifies data issues associated with the Category variable, and a PROC PRINT is specified to display the Category variable's data issues.

PROC FORMAT, DATA Step and PROC PRINT Code:

```
PROC FORMAT LIBRARY=WORK ;
  VALUE $Category_Validation
    'Action'               = 'Action'
    'Action Adventure'     = 'Action Adventure'
    'Action Cops & Robber' = 'Action Cops & Robber'
    'Action Sci-Fi'        = 'Action Sci-Fi'
    'Adventure'            = 'Adventure'
    'Comedy'               = 'Comedy'
    'Drama'                = 'Drama'
    'Drama Mysteries'      = 'Drama Mysteries'
    'Drama Romance'        = 'Drama Romance'
```

```
      'Drama Suspense'       = 'Drama Suspense'
      'Horror'               = 'Horror'
       Other                 = 'ERROR - Invalid Category'
    ;
RUN ;

DATA Validate_Category ;
  SET mydata.Movies_with_Messy_Data ;
  Check_Category = PUT(Category,$Category_Validation.) ;
  IF Check_Category = 'ERROR - Invalid Category' THEN
  DO ;
     PUT 'Category Error: ' Title ;
     OUTPUT ;
  END ;
RUN ;

PROC PRINT DATA=work.Validate_Category
           NOOBS
           N ;
  TITLE "Validation Report for Movie Category Variable" ;
  VAR Category Title Rating Length Studio Year ;
RUN ;
```

SAS Log:

The error messages for the variable, Check_Category, are displayed, below.

```
Category Error: Brave Heart
Category Error: Titanic
Category Error: Forrest Gumpp
Category Error: Christmas Vacatiion
Category Error:
```

Results:

Validation Report for Movie Category Variable

Category	Title	Rating	Length	Studio	Year
Acton Adventure	Brave Heart	R	177	Paramont Pictures	1995
Dramma Romance	Titanic	PG-13	194	Paramount Pictures	1997
Dramma	Forrest Gumpp	PG13	143	Paramont Pictures	1994
Commedy	Christmas Vacatiion	PG-13	97	Warner Brothers	1989
Acton Adventure		R	177	Paramont Pictures	1995
N = 5					

Step #4: Performing Data Transformations

Data transformations are frequently performed by SAS users. From converting a data set structure from wide to long, long to wide, observations to variables, variables to observations, and more, SAS users have a number of choices available to them. A popular procedure used to transform selected variables into observations and observations into variables is the TRANSPOSE procedure. Although PROC TRANSPOSE isn't designed to print or display output, it is handy for restructuring data in a data set, and is typically used in preparation for special types of processing such as, array processing. In its simplest form, data can be transformed with, or without, grouping. In the example, below, an ungrouped transformation is performed on only the numeric variables in the data set.

PROC TRANSPOSE Code:

```
PROC TRANSPOSE DATA=mydata.Movies_with_Messy_Data
               OUT=Movies_Transposed ;
RUN ;
```

Results:

	NAME	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10	COL11	COL12	COL13	COL14	COL15	COL16
1	Length	177	177	103	97	116	130	105	142	143	127	125	127	110	106	98	115
2	Year	1995	1995	1942	1989	1988	1993	1980	1994	1994	1990	1975	1993	1987	1997	1983	1982

	COL17	COL18	COL19	COL20	COL21	COL22	COL23	COL24	COL25	COL26	COL27	COL28	COL29	COL30	COL31
1	120	170	118	124	135	108	101	102	194	120	143	97	98	106	177
2	1976	1983	1991	1977	1989	1984	1939	1939	1997	1976	1994	1989	1983	1997	1995

Data can be restructured with PROC TRANSPOSE using a grouping variable. In the next example, the Movies data set is first sorted in ascending order by the variable RATING, the sort results written to the Movies_Sorted data set, and then the Movies_Sorted data set is transposed using the RATING variable as the by-group variable.

PROC TRANSPOSE Code:

```
PROC SORT DATA=mydata.Movies_with_Messy_Data
          OUT=Movies_Sorted ;
  BY Rating ; /* BY-Group to Transpose */
RUN ;

PROC TRANSPOSE DATA=work.Movies_Sorted
          OUT=Movies_Transposed ;
  VAR Title ; /* Variable to Transpose */
  BY Rating ; /* BY-Group to Transpose */
RUN ;

PROC PRINT DATA=Movies_Transposed ;
RUN ;
```

Results:

	Rating	_NAME_	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	COL9	COL10	COL11
1	G	Title	The Wizard of Oz										
2	GP	Title	The Hunt for Red October										
3	PG	Title	Casablanca	Jaws	Poltergeist	Rocky	Star Wars	Rocky					
4	PG-13	Title	Christmas Vacation	Forrest Gump	Forrest Gump	Ghost	Jurassic Park	Michael	National Lampoon's Vacation	Titanic	Christmas Vacatiion	National Lampoons Vacation	Micheal
5	PG13	Title	Forrest Gumpp										
6	R	Title	Brave Heart	Brave Heart	Coming to America	Dracula	Dressed to Kill	Lethal Weapon	Silence of the Lambs	The Terminator			
7	g	Title	The Wizard of Ozz										
8	r	Title	Scarface										

Step #5: Using Fuzzy Matching Programming Techniques

Fuzzy matching is an essential programming technique used by organizations every day, particularly when the matching variables between data sets are non-existent or unreliable. Although this type of processing can be more involved than traditional matching processing techniques (e.g., interleaving, match-merging, joining, etc.), SAS users have a number of powerful functions available to them, including the Soundex (phonetic matching) algorithm, and the SPEDIS, COMPGED and COMPLEV functions, to help make fuzzy matching easier and more effective to use.

Exploring the Soundex Algorithm

The Soundex (phonetic matching) algorithm involves matching files on words that sound alike. As one of the earliest fuzzy matching techniques, Soundex was invented and patented by Margaret K. Odell and Robert C. Russell in 1918 and 1922 to help match surnames that sound alike. It is limited to finding phonetic matches and adheres to the following rules when performing a search:

- Is case insensitive (ignores case);
- Ignores embedded blanks and punctuations;
- Is better at finding English-sounding names.

Although the Soundex algorithm does a fairly good job with English-sounding names, it often falls-short when dealing with non-English sounding names. In Foley (1999) the author corroborates this by stating, “The Soundex algorithm is not infallible since it has been known to miss similar-sounding surnames like Rogers and Rodgers while matching dissimilar surnames such as Hilbert and Heibronn. “

So, how does the Soundex algorithm work? As implemented, SAS determines whether a name (or a variable’s contents) sounds like another by converting each word to a code. The value assigned to the code consists of the first letter in the word followed by one or more digits. Vowels, A, E, I, O and U, along with H, W, Y, and non-alphabetical characters do not receive a coded value and are ignored; and double letters (e.g., ‘TT’) are assigned a single code value for both letters. The codes derived from each word conform to the letters and values found in the table, below.

Letter	Value
B, P, F, V	1
C, S, G, J, K, Q, X, Z	2
D, T	3
L	4
M, N	5
R	6

To examine how the movie title, Rocky, is assigned a value of R22, R has a value of 6 but is retained as R, O is ignored, C is assigned a value of 2, K is assigned a value of 2, and Y is ignored. The converted code for “Rocky” is then matched with any other name that has the same assigned code. The general syntax of the Soundex algorithm takes the form of:

```
Variable =* “character-string”
```

In the next example, the Soundex algorithm is illustrated using the **=*** operator in a simple DATA step WHERE statement and a PROC SQL WHERE-clause to find similar sounding Movie Titles.

Soundex (=*) Algorithm:

```
DATA Soundex_Matches ;
  SET mydata.Movies_with_Messy_Data ;
  WHERE Title =* "Michael" ;
RUN ;
PROC PRINT DATA=Soundex_Matches NOOBS ;
  TITLE "Soundex Algorithm Matches" ;
RUN ;
```

```
TITLE "Soundex Algorithm Matches" ;
PROC SQL ;
  SELECT *
    FROM mydata.Movies_with_Messy_Data
      WHERE Title =* "Michael" ;
QUIT ;
```

Results:

Soundex Algorithm Matches

Title	Length	Category	Year	Studio	Rating
Michael	106	Drama	1997	Warner Bros	PG-13
Micheal	106	Drama	1997	Warner Brothers	PG-13

Exploring the SPEDIS Function

The SPEDIS, Spelling Distance, function and its two arguments evaluates possible matching scenarios by translating a keyword into a query containing the smallest distance value. Because the SPEDIS function evaluates numerous scenarios, it can experience varying performance issues in comparison to other matching techniques. The SPEDIS function evaluates query and keyword arguments returning non-negative spelling distance values. A derived value of zero indicates an exact match. Generally, derived values are less than 100, but, on occasion, can exceed 200. Users can control the matching process by specifying spelling distance values greater than zero (e.g., 10, 20, etc.).

So, how does the SPEDIS function work? As implemented, SAS determines whether a name (or a variable's contents) is alike by computing an asymmetric spelling distance between two words. The SPEDIS function computes the costs associated with converting the keyword to the query, as illustrated in the following table, below.

Operation	Cost	Description
Match	**0**	**No change**
Singlet	**25**	**Delete one of a double letter**
Doublet	**50**	**Double a letter**
Swap	**50**	**Reverse the order of two consecutive letters**
Truncate	**50**	**Delete a letter from the end**
Append	**35**	**Add a letter to the end**
Delete	**50**	**Delete a letter from the middle**
Insert	**100**	**Insert a letter in the middle**
Replace	**100**	**Replace a letter in the middle**
Firstdel	**100**	**Delete the first letter**
Firstins	**200**	**Insert a letter at the beginning**
Firstrep	**200**	**Replace the first letter**

The general syntax of the SPEDIS function takes the form of:

SPEDIS (query, keyword)

In this first example, a simple DATA step with a WHERE statement and a PROC SQL with a WHERE-clause are illustrated to show the values derived by the SPEDIS function for finding exact matches for the Movie Title, “Michael”.

SPEDIS Function:

```
DATA work.SPEDIS_Matching ;
 SET mydata.Movies_with_Messy_Data ;
 Spedis_Value = SPEDIS(Title,"Michael") ;
RUN ;
PROC PRINT DATA=work.SPEDIS_Matching
           NOOBS ;
 TITLE "SPEDIS Function Matches" ;
 WHERE SPEDIS(Title,"Michael") GE 0 ;
RUN ;
```

```
TITLE "SPEDIS Function Matches" ;
PROC SQL ;
 SELECT *,
        SPEDIS(Title,"Michael")
               AS Spedis_Value
  FROM mydata.Movies_with_Messy_Data
   WHERE SPEDIS(Title,"Michael") GE 0 ;
QUIT ;
```

Results:

SPEDIS Function Matches

Title	Length	Category	Year	Studio	Rating	Spedis_Value
Brave Heart	177	Action Adventure	1995	Paramount Pictures	R	76
Brave Heart	177	Acton Adventure	1995	Paramont Pictures	R	76
Casablanca	103	Drama	1942	MGM / UA	PG	75
Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13	63
Coming to America	116	Comedy	1988	Paramount Pictures	R	67
Dracula	130	Horror	1993	Columbia TriStar	R	100
Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R	65
Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13	77
Forrest Gump	143	Drama	1994	Paramount Pictures	PG-13	77
Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13	120
Jaws	125	Action Adventure	1975	Universal Studios	PG	137
Jurassic Park	127	Action	1993	Universal Pictures	PG-13	73
Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R	58
National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13	48
Poltergeist	115	Horror	1982	MGM / UA	PG	80
Rocky	120	Action Adventure	1976	MGM / UA	PG	120
Scarface	170	Action Cops & Robber	1983	Universal Studios	r	87
Silence of the Lambs	118	Drama Suspense	1991	Orion	R	55
Star Wars	124	Action Sci-Fi	1977	Lucas Film Ltd	PG	96
The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	GP	56
The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R	67
The Wizard of Oz	101	Adventure	1939	MGM / UA	G	66
The Wizard of Ozz	102	Adventure	1939	MGM - UA	g	64
Titanic	194	Dramma Romance	1997	Paramount Pictures	PG-13	92
Rocky	120	Action Adventure	1976	MGM / UA	PG	120
Forrest Gumpp	143	Dramma	1994	Paramont Pictures	PG13	73
Christmas Vacation	97	Commedy	1989	Warner Brothers	PG-13	62
National Lampoons Vacation	98	Comedy	1983	Warner Brothers	PG-13	49
Micheal	106	Drama	1997	Warner Brothers	PG-13	7
	177	Acton Adventure	1995	Paramont Pictures	R	400

In the next example, a simple DATA step with a WHERE statement and a PROC SQL with a WHERE-clause are illustrated to how the SPEDIS function is used to find exact matches for Movie Titles.

SPEDIS Function:

```
DATA work.SPEDIS_Matching ;
  SET mydata.Movies_with_Messy_Data ;
  Spedis_Value = SPEDIS(Title,"Michael") ;
RUN ;
PROC PRINT DATA=work.SPEDIS_Matching
           NOOBS ;
  TITLE "SPEDIS Function Matches" ;
  WHERE SPEDIS(Title,"Michael") = 0 ;
RUN ;
```

```
TITLE "SPEDIS Function Matches" ;
PROC SQL ;
 SELECT *,
        SPEDIS(Title,"Michael")
               AS Spedis_Value
  FROM mydata.Movies_with_Messy_Data
   WHERE SPEDIS(Title,"Michael") = 0 ;
QUIT ;
```

Results:

SPEDIS Function Matches

Title	Length	Category	Year	Studio	Rating	Spedis_Value
Michael	106	Drama	1997	Warner Bros	PG-13	0

In the next example, a DATA step with a WHERE statement and a PROC SQL with a WHERE-clause are illustrated to show how the SPEDIS function is used to find spelling variations associated with Movie Titles.

SPEDIS Function:

```
DATA work.SPEDIS_Matching ;
  SET mydata.Movies_with_Messy_Data ;
  Spedis_Value = SPEDIS(Title,"Michael") ;
RUN ;
PROC PRINT DATA=work.SPEDIS_Matching
           NOOBS ;
  TITLE "SPEDIS Function Matches" ;
  WHERE SPEDIS(Title,"Michael") LE 20 ;
RUN ;
```

```
TITLE "SPEDIS Function Matches" ;
PROC SQL ;
 SELECT *,
        SPEDIS(Title,"Michael")
               AS Spedis_Value
  FROM mydata.Movies_with_Messy_Data
   WHERE SPEDIS(Title,"Michael") LE 20 ;
QUIT ;
```

Results:

SPEDIS Function Matches

Title	Length	Category	Year	Studio	Rating	Spedis_Value
Michael	106	Drama	1997	Warner Bros	PG-13	0
Micheal	106	Drama	1997	Warner Brothers	PG-13	7

Exploring the COMPGED Function

The COMPGED function is another fuzzy matching technique used by the SAS user community. It works by computing and using a Generalized Edit Distance (GED) score when comparing two text strings. In Teres (2011), the author describes the Generalized Edit Distance score as "a generalization of the Levenshtein edit distance, which is a measure of dissimilarity between two strings." Sloan and Hoicowitz describe their experience using the COMPGED function to match data sets with unreliable identifiers (or keys) by pointing out, "The higher the GED score the less likely the two strings match." Conversely, for the greatest likelihood of a match with the COMPGED function users should seek the lowest derived score from evaluating all the possible ways of matching string-1 with string-2.

The COMPGED function returns values that are multiples of 10, e.g., 20, 100, 200, etc. In Cadieux and Bretheim's (2014) paper, the authors mention that most COMPGED scores of 100 or less are valid matches. So how is the COMPGED function used to compare two text strings for possible matching? The general syntax of the COMPGED function takes the form of:

COMPGED (string-1, string-2 <, cutoff-value> <, modifier>)

Required Arguments:

string-1 specifies a character variable, constant or expression.
string-2 specifies a character variable, constant or expression.

Optional Arguments:

cutoff-value specifies a numeric variable, constant or expression. If the actual generalized edit distance is greater than the value of ***cutoff***, the value that is returned is equal to the value of ***cutoff***.

modifier specifies a value that alters the action of the COMPGED function. Valid modifier values are:

- i or I Ignores the case in string-1 and string-2.
- l or L Removes leading blanks before comparing the values in string-1 or string-2.
- n or N Ignores quotation marks around string-1 or string-2.
- : (colon) Truncates the longer of string-1 or string-2 to the length of the shorter string.

In this first example, a PROC SQL inner join is constructed along with the specification of a COMPGED function to allow for matches that are not perfect. The COMPGED function derives a value corresponding to the computed generalized edit distance (GED) score as, COMPGED_Score in the new table, Movies_Fuzzy_Matches. As illustrated in the results, the COMPGED_Score column contains a subsetted value between 0 and 100 due to the "cutoff-value" of 100 as is specified in the WHERE-clause expression. Along with the "cutoff-value, the WHERE-clause also eliminates missing titles from further consideration.

PROC SQL Join with COMPGED Function:

```
proc sql noprint ;
 create table Movies_Fuzzy_Matches as
  select M.Title,
         Rating,
         Category,
         Actor_Leading,
         Actor_Supporting,
         COMPGED(M.Title,A.Title) AS COMPGED_Score
    from mydata.Movies_with_Messy_Data M,
         mydata.Actors_with_Messy_Data A
     where M.Title NE "" AND
           CALCULATED COMPGED_Score LE 100
      order by M.Title ;
quit ;
```

Results:

	Title	Rating	Category	Actor_Leading	Actor_Supporting	COMPGED_Score
1	Brave Heart	R	Acton Adventure	Mel Gibson	Sophie Marceau	0
2	Brave Heart	R	Action Adventure	Mel Gibson	Sophie Marceau	0
3	Coming to America	R	Comedy	Eddie Murphy	Arsenio Hall	0
4	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
5	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
6	Forrest Gumpp	PG13	Dramma	Tom Hanks	Sally Field	20
7	Lethal Weapon	R	Action Cops & Robber	Mel Gibson	Danny Glover	0
8	Michael	PG-13	Drama	John Travolta	Andie MacDowell	0
9	Micheal	PG-13	Drama	John Travolta	Andie MacDowell	20
10	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
11	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
12	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
13	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
14	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
15	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
16	Silence of the Lambs	R	Drama Suspense	Anthony Hopkins	Jodie Foster	0
17	The Terminator	R	Action Sci-Fi	Arnold Schwarzenegge	Michael Biehn	0
18	Titanic	PG-13	Dramma Romance	Leonardo DiCaprio	Kate Winslet	0

In the next example, the "cutoff-value" is maintained at 100, as it was in the previous example. In addition to the COMPGED function, a modifier value of "I" has been specified to tell SAS to ignore the case of both string-1 and string-2. Unlike the previous example's results, the results for this example show that the row associated with the movie "Ghost" in the argument for string-1 matches the value of "GHOST" in the argument for string-2.

PROC SQL Join with COMPGED Function and a Modifier of 'I':

```
proc sql noprint ;
 create table Movies_Fuzzy_Matches as
  select M.Title,
         Rating,
         Category,
         Actor_Leading,
         Actor_Supporting,
         COMPGED(M.Title,A.Title,'I') AS COMPGED_Score
    from mydata.Movies_with_Messy_Data M,
         mydata.Actors_with_Messy_Data A
     where M.Title NE "" AND
           CALCULATED COMPGED_Score LE 100
      order by M.Title ;
quit ;
```

Results:

	Title	Rating	Category	Actor_Leading	Actor_Supporting	COMPGED_Score
1	Brave Heart	R	Action Adventure	Mel Gibson	Sophie Marceau	0
2	Brave Heart	R	Acton Adventure	Mel Gibson	Sophie Marceau	0
3	Coming to America	R	Comedy	Eddie Murphy	Arsenio Hall	0
4	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
5	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
6	Forrest Gumpp	PG13	Dramma	Tom Hanks	Sally Field	20
7	Ghost	PG-13	Drama Romance	Patrick Swayze	Demi Moore	0
8	Lethal Weapon	R	Action Cops & Robber	Mel Gibson	Danny Glover	0
9	Michael	PG-13	Drama	John Travolta	Andie MacDowell	0
10	Micheal	PG-13	Drama	John Travolta	Andie MacDowell	20
11	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
12	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
13	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
14	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
15	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
16	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
17	Silence of the Lambs	R	Drama Suspense	Anthony Hopkins	Jodie Foster	0
18	The Hunt for Red October	GP	Action Adventure	Sean Connery	Alec Baldwin	100
19	The Terminator	R	Action Sci-Fi	Arnold Schwarzenegge	Michael Biehn	0
20	Titanic	PG-13	Dramma Romance	Leonardo DiCaprio	Kate Winslet	0

In the next example, the COMPGED function's modifier value of "I" has been removed and the "cutoff-value" was increased from 100 to 400. By increasing the "cutoff-value", we liberalized the matching process to perform matches when the matching columns are not perfect. Unlike the previous example where the modifier value of "I" was specified, the results for this example shows the row associated with the movie "Ghost" with a COMPGED_Score of 400, and the argument for string-1 matches the value of "GHOST" in the argument for string-2,.

PROC SQL Join with COMPGED Function and COMPGED_Score LE 400:

```
proc sql noprint ;
 create table Movies_Fuzzy_Matches as
  select M.Title,
         Rating,
         Category,
         Actor_Leading,
         Actor_Supporting,
         COMPGED(M.Title,A.Title) AS COMPGED_Score
    from mydata.Movies_with_Messy_Data M,
         mydata.Actors_with_Messy_Data A

     where M.Title NE "" AND
           CALCULATED COMPGED_Score LE 400
      order by M.Title ;
quit ;
```

Results:

	Title	Rating	Category	Actor_Leading	Actor_Supporting	COMPGED_Score
1	Brave Heart	R	Action Adventure	Mel Gibson	Sophie Marceau	0
2	Brave Heart	R	Acton Adventure	Mel Gibson	Sophie Marceau	0
3	Coming to America	R	Comedy	Eddie Murphy	Arsenio Hall	0
4	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
5	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
6	Forrest Gumpp	PG13	Dramma	Tom Hanks	Sally Field	20
7	Ghost	PG-13	Drama Romance	Patrick Swayze	Demi Moore	400
8	Lethal Weapon	R	Action Cops & Robber	Mel Gibson	Danny Glover	0
9	Michael	PG-13	Drama	John Travolta	Andie MacDowell	0
10	Micheal	PG-13	Drama	John Travolta	Andie MacDowell	20
11	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
12	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
13	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
14	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
15	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
16	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
17	Silence of the Lambs	R	Drama Suspense	Anthony Hopkins	Jodie Foster	0
18	The Hunt for Red October	GP	Action Adventure	Sean Connery	Alec Baldwin	100
19	The Terminator	R	Action Sci-Fi	Arnold Schwarzenegge	Michael Biehn	0
20	Titanic	PG-13	Dramma Romance	Leonardo DiCaprio	Kate Winslet	0

In the next example, the COMPGED function has a "cutoff-value" for the COMPGED_Score set at 100, and a modifier value of "INL" to ignore the case, remove leading blanks, and ignore quotes around string-1 and string-2. As before, the results for this example show the row associated with the movie "Ghost" in the argument for string-1 matches the value of "GHOST" in the argument for string-2.

PROC SQL Join with COMPGED Function and Modifier of 'INL':

```
proc sql noprint ;
 create table Movies_Fuzzy_Matches as
  select M.Title,
         Rating,
         Category,
         Actor_Leading,
         Actor_Supporting,
         COMPGED(M.Title,A.Title,'INL') AS COMPGED_Score
    from mydata.Movies_with_Messy_Data M,
         mydata.Actors_with_Messy_Data A
     where M.Title NE "" AND
           CALCULATED COMPGED_Score LE 100
      order by M.Title ;
quit ;
```

Results:

	Title	Rating	Category	Actor_Leading	Actor_Supporting	COMPGED_Score
1	Brave Heart	R	Action Adventure	Mel Gibson	Sophie Marceau	0
2	Brave Heart	R	Acton Adventure	Mel Gibson	Sophie Marceau	0
3	Coming to America	R	Comedy	Eddie Murphy	Arsenio Hall	0
4	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
5	Forrest Gump	PG-13	Drama	Tom Hanks	Sally Field	0
6	Forrest Gumpp	PG13	Dramma	Tom Hanks	Sally Field	20
7	Ghost	PG-13	Drama Romance	Patrick Swayze	Demi Moore	0
8	Lethal Weapon	R	Action Cops & Robber	Mel Gibson	Danny Glover	0
9	Michael	PG-13	Drama	John Travolta	Andie MacDowell	0
10	Micheal	PG-13	Drama	John Travolta	Andie MacDowell	20
11	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
12	National Lampoon's Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
13	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	30
14	National Lampoons Vacation	PG-13	Comedy	Chevy Chase	Beverly D'Angelo	0
15	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
16	Rocky	PG	Action Adventure	Sylvester Stallone	Talia Shire	0
17	Silence of the Lambs	R	Drama Suspense	Anthony Hopkins	Jodie Foster	0
18	The Hunt for Red October	GP	Action Adventure	Sean Connery	Alec Baldwin	100
19	The Terminator	R	Action Sci-Fi	Arnold Schwarzenegge	Michael Biehn	0
20	Titanic	PG-13	Dramma Romance	Leonardo DiCaprio	Kate Winslet	0

Exploring the COMPLEV Function

The COMPLEV function is another fuzzy matching technique used by the SAS user community. It stands for Levenshtein Edit Distance. As with the SPEDIS and COMPGED functions, the COMPLEV function provides an indicator of how close two strings are, with one exception. In lieu of assigning a score for each operation, it returns the number of operations. The general syntax of the COMPLEV function takes the form of:

COMPLEV (string-1, string-2 <, cutoff-value> <, modifier>)

Required Arguments:

string-1 specifies a character variable, constant or expression.

string-2 specifies a character variable, constant or expression.

Optional Arguments:

cutoff-value specifies a numeric variable, constant or expression. If the actual Levenshtein edit distance is greater than the value of *cutoff*, the value that is returned is equal to the value of ***cutoff***.

modifier specifies a value that alters the action of the COMPLEV function. Valid modifier values are:

- i or I Ignores the case in string-1 and string-2.
- l or L Removes leading blanks before comparing the values in string-1 or string-2.
- n or N Ignores quotation marks around string-1 or string-2.
- : (colon) Truncates the longer of string-1 or string-2 to the length of the shorter string.

In the next example, a PROC SQL inner join is constructed along with the specification of a COMPLEV function to determine the best possible match producing a value of, COMPLEV_Number. As illustrated in the results, the COMPLEV_Number column displays the number of operations that have been performed. The lower the value the better the match (e.g., 0 = Best match, 1 = Next Best match, etc.).

PROC SQL Join with COMPLEV Function:

```
proc sql ;
  select M.Title,
         Rating,
         Length,
         Category,
         COMPLEV(M.Category,"Drama") AS COMPLEV_Number
    from mydata.Movies_with_Messy_Data M
     where M.Title NE ""
      order by M.Title ;
quit ;
```

Results:

	Title	Rating	Length	Category	COMPLEV_Number
1	Brave Heart	R	177	Action Adventure	16
2	Brave Heart	R	177	Acton Adventure	15
3	Casablanca	PG	103	Drama	0
4	Christmas Vacatiion	PG-13	97	Commedy	6
5	Christmas Vacation	PG-13	97	Comedy	6
6	Coming to America	R	116	Comedy	6
7	Dracula	R	130	Horror	5
8	Dressed to Kill	R	105	Drama Mysteries	10
9	Forrest Gump	PG-13	143	Drama	0
10	Forrest Gump	PG-13	142	Drama	0
11	Forrest Gumpp	PG13	143	Dramma	1
12	Ghost	PG-13	127	Drama Romance	8
13	Jaws	PG	125	Action Adventure	16
14	Jurassic Park	PG-13	127	Action	6
15	Lethal Weapon	R	110	Action Cops & Robber	20
16	Michael	PG-13	106	Drama	0
17	Micheal	PG-13	106	Drama	0
18	National Lampoon's Vacation	PG-13	98	Comedy	6
19	National Lampoons Vacation	PG-13	98	Comedy	6
20	Poltergeist	PG	115	Horror	5
21	Rocky	PG	120	Action Adventure	16
22	Rocky	PG	120	Action Adventure	16
23	Scarface	r	170	Action Cops & Robber	20
24	Silence of the Lambs	R	118	Drama Suspense	9
25	Star Wars	PG	124	Action Sci-Fi	13
26	The Hunt for Red October	GP	135	Action Adventure	16
27	The Terminator	R	108	Action Sci-Fi	13
28	The Wizard of Oz	G	101	Adventure	9
29	The Wizard of Ozz	g	102	Adventure	9
30	Titanic	PG-13	194	Dramma Romance	9

In the next example, we modify what was produced previously and restrict our PROC SQL WHERE-clause to subset non-missing Titles and COMPLEV_Number values containing either 0 or 1. The results confirm that a fuzzy matching process using the COMPLEV function to select values of 0 or 1, representing the "best" matches for COMPLEV_Number, has been correctly performed.

PROC SQL Join with COMPLEV Function:

```
proc sql ;
  title "COMPLEV Function Matches" ;
  select M.Title,
         Rating,
         Length,
         Category,
         COMPLEV(M.Category,"Drama") AS COMPLEV_Number
    from mydata.Movies_with_Messy_Data M
     where M.Title NE ""
       AND COMPLEV_Number LE 1
      order by M.Title ;
quit ;
```

Results:

	Title	Rating	Length	Category	COMPLEV_Number
1	Casablanca	PG	103	Drama	0
2	Forrest Gump	PG-13	143	Drama	0
3	Forrest Gump	PG-13	142	Drama	0
4	Forrest Gumpp	PG13	143	Dramma	1
5	Michael	PG-13	106	Drama	0
6	Micheal	PG-13	106	Drama	0

Conclusion

When data originating from multiple sources contain duplicate observations, duplicate and/or unreliable keys, missing values, invalid values, capitalization and punctuation issues, inconsistent matching variables, and imprecise text identifiers, the matching process is often compromised by unreliable and/or unpredictable results. This paper demonstrates a five-step approach including identifying, cleaning and standardizing data irregularities, conducting data transformations, and utilizing special-purpose programming techniques such as the application of SAS functions, the SOUNDEX algorithm, the SPEDIS function, approximate string matching functions including COMPGED and COMPLEV, and an assortment of constructive programming techniques to standardize and combine data sets together when the matching columns are unreliable or less than perfect.

References

Cadieux, Richard and Daniel R. Brethiem (2014). *"Matching Rules: Too Loose, Too Tight, or Just Right?"*, Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Cody, Ron (2017). *"Cody's Data Cleaning Techniques Using SAS®, Third Edition"*, SAS Press, SAS Institute, Cary, NC, USA.

Dunn, Toby (2014). *"Getting the Warm and Fuzzy Feeling with Inexact Matching"*, Proceedings of the 2014 SAS Global Forum (SGF) Conference.

Foley, Malachy J. (1999). *"Fuzzy Merges: Examples and Techniques"*, Proceedings of the 1999 SAS Users Group International (SUGI) Conference.

Lafler, Kirk Paul and Stephen Sloan (2017). *"A Quick Look at Fuzzy Matching Programming Techniques Using SAS® Software"*, Proceedings of the 2017 Western Users of SAS Software (WUSS) Conference.

Lafler, Kirk Paul (2016). *"Removing Duplicates Using SAS®"*, Proceedings of the 2016 MidWest SAS Users Group (MWSUG) Conference.

Patridge, Charles (1997). *"The Fuzzy Feeling SAS Provides: Electronic Matching of Records without Common Keys",* Proceedings of the 1997 SAS Users Group International (SUGI) Conference.

Russell, Kevin (January 27, 2015). *"How to Perform a Fuzzy Match Using SAS Functions".* blogs.sas.com.

Roesch, Amanda (2012). *"Matching Data Using Sounds-Like Operators and SAS® Compare Functions",* Proceedings of the 2012 SAS Global Forum (SGF) Conference.

Sloan, Stephen and Dan Hoicowitz (2016). *"Fuzzy Matching: Where Is It Appropriate and How Is It Done? SAS Can Help.",* Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Staum, Paulette (2007). *"Fuzzy Matching using the COMPGED Function",* Proceedings of the 2007 NorthEast SAS Users Group (NESUG) Conference.

Teres, Jedediah J. (2011). *"Using SQL Joins to Perform Fuzzy Matches on Multiple Identifiers",* Proceedings of the 2011 NorthEast SAS Users Group (NESUG) Conference.

"Transforming SAS Data Sets", (2000). http://www.rhoworld.com/pdf/ch599.pdf.

Zirbel, Douglas (2009). *"Learn the Basics of PROC TRANSPOSE",* Proceedings of the 2009 SAS Global Forum (SGF) Conference.

Acknowledgments

The authors thank Clarence Jackson and Greg Gengo, SouthCentral SAS Users Group (SCSUG) Conference Co-Chairs for accepting our abstract and paper; the SouthCentral SAS Users Group (SCSUG) Executive Board; and SAS Institute for organizing and supporting a great conference!

Trademark Citations

About the Authors

Kirk Paul Lafler is an entrepreneur, consultant and founder of Software Intelligence Corporation, and has been using SAS since 1979. Kirk is a SAS application developer, programmer, certified professional, provider of IT consulting services, mentor, advisor, professor at UC San Diego Extension and educator to SAS users around the world, and emeritus sasCommunity.org Advisory Board member. As the author of six books including Google® Search Complete (Odyssey Press. 2014) and PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press. 2013); Kirk has written hundreds of papers and articles; been an Invited speaker and trainer at hundreds of SAS International, regional, special-interest, local, and in-house user group conferences and meetings; and is the recipient of 25 "Best" contributed paper, hands-on workshop (HOW), and poster awards.

Stephen Sloan has worked at Accenture in the Services, Consulting, and Digital groups and is currently a senior manager in the SAS Analytics area. He has worked in a variety of functional areas including Project Management, Data Management, and Statistical Analysis. Stephen has had the good fortune to have worked with many talented people at SAS Institute. Stephen has a B.A. in Mathematics from Brandeis University, M.S. degrees in Mathematics and Computer Science from Northern Illinois University, and an MBA from Stern Business School at New York University.

Comments and suggestions may be sent to:

Kirk Paul Lafler
SAS® Consultant, Application Developer, Programmer, Data Analyst, Educator and Author
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: http://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd

~ ~ ~ ~ ~ ~ ~

Stephen Sloan
Senior Manager in SAS Analytics
Accenture
E-mail: Stephen.B.Sloan@accenture.com