Data Cleaning Using SAS Programming

Boston SAS Users Group SAS Blowout
October 18, 2024

Jacqueline Johnson, DrPH
Principal Analytical Training Consultant, SAS

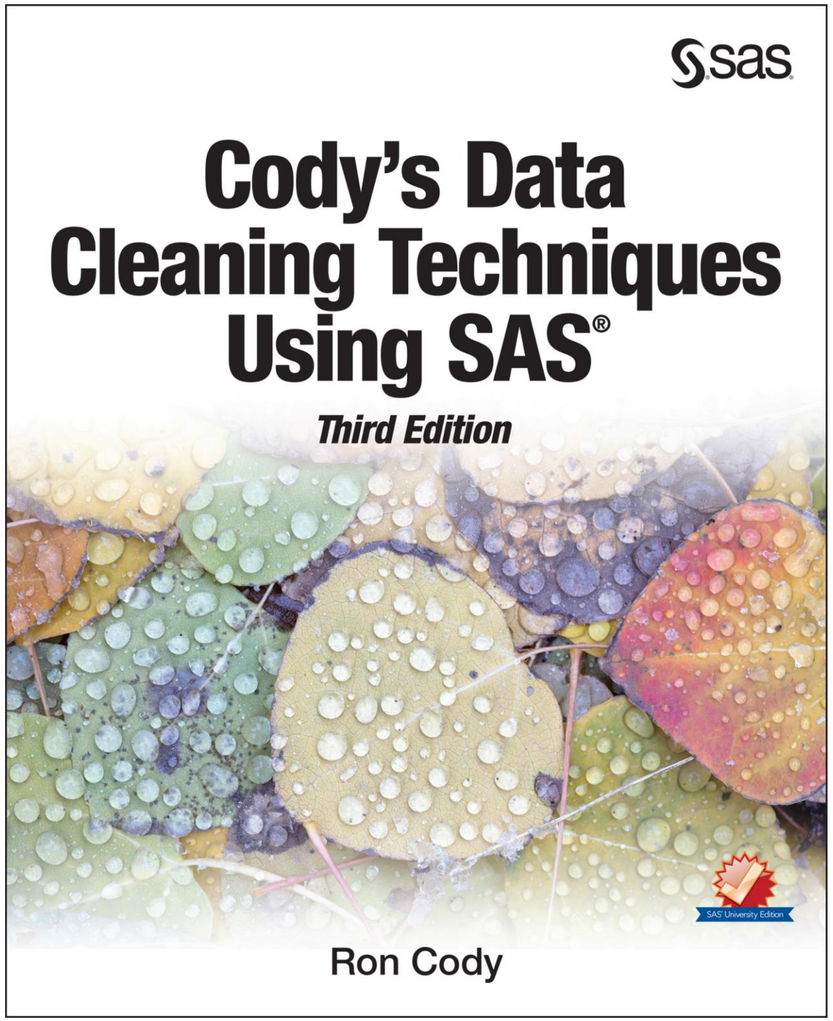
Jacqueline.Johnson@sas.com

Outline

- Using Perl regular expressions to detect data errors in character variables.
- Using SAS formats to standardize data.
- Creating integrity constraints to restrict data values allowed in a data set.

Reference

SAS Press Book



- Webinar material comes from this book!
- Available on Redshelf and Amazon.
- Programs and datasets are downloadable for free from the Ron Cody SAS Author Page. Includes several helpful macros!

Part 1- Perl Regular Expressions

- Regular expressions can be used to describe text patterns.
- A regular expression starts and ends with a delimiter, the most common being a forward slash (/).
- For example, the expression /cat/ will match the word "cat".
- The power of a regular expression is that there are meta-characters that can reference classes of characters, such as all digits or all upper- and lowercase letters.
- You can use regular expressions in SAS functions to verify if a string complies with a particular pattern.

Some Common Expressions

Regular Expression	What it Matches
\d	Any digit
\D	Any non-digit (Expressions are case sensitive)
\s	Whitespace character (blank, tab, line feed, etc.)
\b	Word boundary (blank, beginning or end of string)
\w	Word character (letter or _)
^	Beginning of a string
$	End of a string
[abc]	An 'a' or a 'b' or a 'c'
[0-9]	Digits 0 to 9
High|Low	The string 'High' or 'Low'; the | means 'or'

Some Examples of Regular Expressions

Expression	String	Result
/\d\d\d/	123	Match at position 1
	12345	Match at position 1
	abc888xyz	Match at position 4
/^\d\d\d/	abc888xyz	No match. ^ means start at the beginning of the line
/\(\d\d\d\)/	(800)	Match at position 1

In the last example, you need to precede the open and closed parentheses with a \ because parentheses in a regular expression have another meaning (grouping) and the \ before either '(' or ')' means to treat the character as a parenthesis, not a grouping character.

Repetition Operators

{*n*}	Matches previous expression *n* times.
{*n*,*m*}	Matches previous expression at least *n* times and not more than *m* times.
*	Matches previous expression zero or more times.
+	Matches previous expression one or more times.
?	Matches previous expression zero or one time.

Examples:

/\d{4}/ matches four digits

/\d{4,6}/ matches between four and six digits

/cat*/ matches "ca" followed by 0 or more "t's"

/cat?/ matches "ca" followed by 0 or 1 "t"

/c(at)?/ matches "c" followed by zero or one occurrences of "at"

/\d\d+/ matches one or more digits

Expression Example – US and Canada Phone Numbers

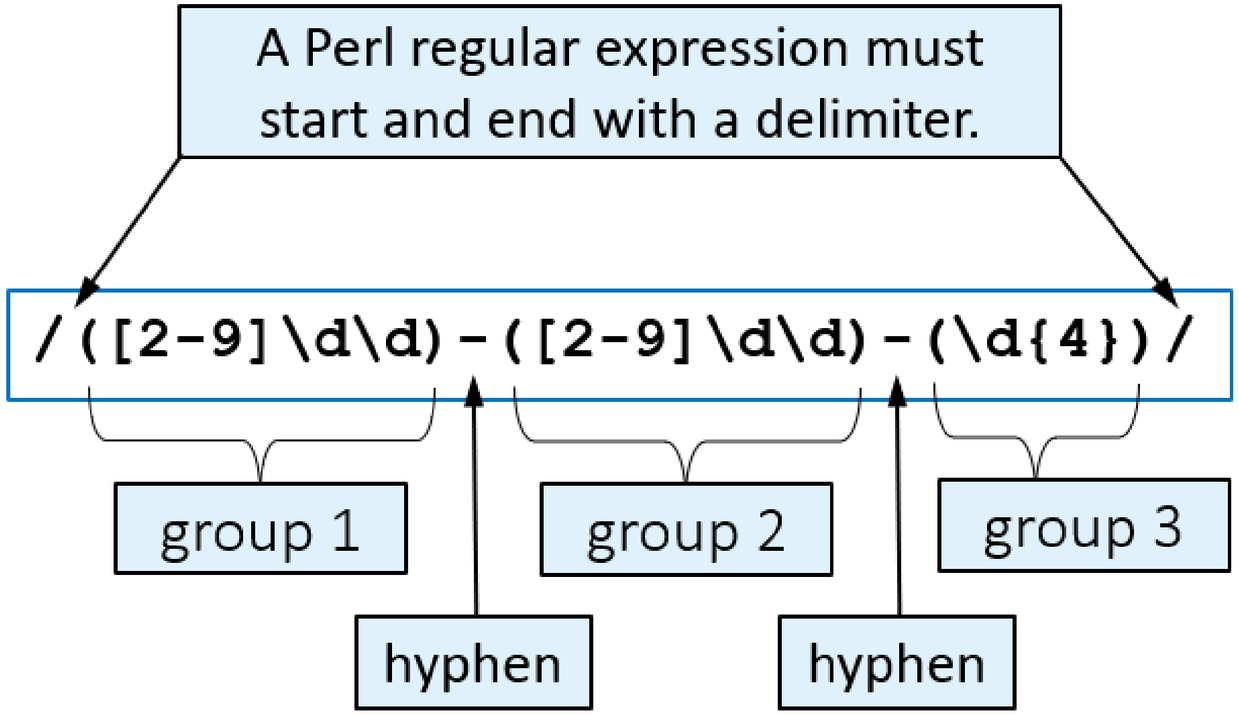
Phone
123-456-7890
234-567-8901
1-345-678-9012
567-8901
789-012-3456
890-123-456
901-234-5678 US

Return phone numbers following these rules:

- Contains a three-digit area code, then a hyphen, then a three-digit prefix, then another hyphen, and a four-digit line number.
- The first digit of the area code and the prefix cannot start with a 0 or 1.

Expression Example – US and Canada Phone Numbers

Phone
123-456-7890
234-567-8901
1-345-678-9012
567-8901
789-012-3456
890-123-456
901-234-5678 US



Expression Example

Which Perl regular expression will *not* find the values EF3, EF-3, EF4, and EF-4?

- ✓ `'/(EF3|EF-3|EF4|EF-4)/'`
- ✓ `'/(EF-?3|EF-?4)/'`
- ✓ `'/EF-?(3|4)/'`
- ✓ `'/EF-?[34]/'`
- ✗ `'/EF.[34]/'`

(…)	Parentheses are for grouping.
|	Vertical line is for OR situation.
?	Matches the preceding character 0 or 1 times.
[…]	Matches a character in the brackets.
.	Matches any character.

Testing Expressions with the PRXMATCH Function

PRXMATCH (*Regular Expression*, *String*);

- *Regular-Expression* is a regular expression
  - Either an expression in quotation marks or the name of a character variable that represents the expression.
- *String* is the character value that you are testing.
- If a match is found, the function returns the value of the starting position in the string.
- If a match is not found, the function returns a 0.

Example – Checking DX

DX should be three numbers, a decimal, then three numbers

Patients Dataset
First 10 Rows

Patno	Dx
	195.920
001	713.410
002	047.570
003	108.510
004	669.860
005	078.160
005	078.160
006	967.570
007	640.260
007	564.870

```
title1 "Errors in DX Values";
proc print data=clean.patients(keep=patno dx);
where prxmatch("/\d\d\d\.\d\d\d/",Dx) = 0;
run;
```

Errors in DX Values

Obs	Patno	Dx
13	011	530.abc
91	091	
94	094	V23.000

Example – Checking Zip Codes

Zip should be 5 digits or 5 digits, a dash, then 4 digits

Zip Codes

Zip
12345
78010-5049
12Z44
ABCDE
08822

```
title1 "Errors in Zip";
proc print data=zip noobs;
where prxmatch("/\d{5}(-\d{4})?/",Zip)=0;
run;
```

Errors in Zip

Zip
12Z44
ABCDE

Part 2 – Data Standardization

Listing of Company

Name
International Business Machines
International Business Macnines, Inc.
IBM
Little and Sons
Little & Sons
Little and Son
MacHenrys
McHenrys
MacHenries
McHenry's
Harley Davidson

Name is entered in several different forms that mean the same company

Formatted values we want to use

Using a Format to Standardize Values

```
proc format;
   value $Company
      "International Business Machines, Inc." =
      "International Business Machines"
      "IBM" = "International Business Machines"
      "Little & Sons" = "Little and Sons"
      "Little and Son" = "Little and Sons"
      "MacHenrys"      = "McHenrys"
      "MacHenries"     = "McHenrys"
      "McHenry's"      = "McHenrys";
run;
```

Standard values are to the right of the = sign

Using a PUT Function to Create a Formatted Variable

```
data Standard;
    set Company;
    Standard_Name = put(Name,$Company.);
run;
```

Listing of Standard

Name	Standard_Name
International Business Machines	International Business Machines
International Business Macnines, Inc.	International Business Macnines
IBM	International Business Machines
Little and Sons	Little and Sons
Little & Sons	Little and Sons
Little and Son	Little and Sons
MacHenrys	McHenrys
McHenrys	McHenrys
MacHenries	McHenrys
McHenry's	McHenrys
Harley Davidson	Harley Davidson

Create a Format From a SAS Dataset

- Typing formatted values directly into PROC FORMAT code can be time consuming.
- Instead, type the values elsewhere then create a SAS dataset from the file.
- Example:
  - Create an Excel file with the desired changes
  - Create a SAS dataset from the Excel file using PROC IMPORT or the XLSX libname engine

Create the Initial SAS Dataset

```
proc import datafile="c:/Company_Standards.xlsx"
            dbms=xlsx
            out=work.standard replace;
run;
```

Listing of Standard

Name	Standard_Name
International Business Machines	International Business Machines
International Business Macnines, Inc.	International Business Macnines
IBM	International Business Machines
Little and Sons	Little and Sons
Little & Sons	Little and Sons
Little and Son	Little and Sons
MacHenrys	McHenrys
McHenrys	McHenrys
MacHenries	McHenrys
McHenry's	McHenrys
Harley Davidson	Harley Davidson

Add Control Variables to the Dataset

- Start – The starting value in a range. If there is only one value (as in this case), you do not have to include a value for End in the control data set.
- End – This is an ending value if you have a range such as 10 to 20 (with 10 being the Start value and 20 being the end value).
- Label – This is the format label.
- Fmtname – This is the name of the format that you want to create. Do not include a dollar sign in the name, even if this is a character format.
- Type – Use a 'C' if you are creating a character format; use an 'N' if you are creating a numeric format.

Creating the Control Dataset

Listing of Standard

Name	Standard_Name
International Business Machines	International Business Machines
International Business Macnines, Inc.	International Business Macnines
IBM	International Business Machines
Little and Sons	Little and Sons
Little & Sons	Little and Sons
Little and Son	Little and Sons
MacHenrys	McHenrys
McHenrys	McHenrys
MacHenries	McHenrys
McHenry's	McHenrys
Harley Davidson	Harley Davidson

```
data Control;
   set Standard
      (rename=(Name=Start
               Standard_Name=Label));
   retain Fmtname "Company" Type "C";
run;
```

Listing of Data Set Control

Start	Label	Fmtname	Type
International Business Machines, Inc.	International Business Machines	Company	C
IBM	International Business Machines	Company	C
Little & Sons	Little and Sons	Company	C
Little and Son	Little and Sons	Company	C
MacHenrys	McHenrys	Company	C
MacHenries	McHenrys	Company	C
McHenry's	McHenrys	Company	C

Creating the New Format: $Company.

```
proc format library=work
     cntlin=Control fmtlib;
run;
```

```
113         proc format library=work
114              cntlin=Control fmtlib;
NOTE: Format $COMPANY has been output.
115         run;
```

FORMAT NAME: $COMPANY LENGTH: 31 NUMBER OF VALUES: 7		
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 31 FUZZ: 0		
START	END	LABEL (VER. V7|V8 15OCT2024:21:48:01)
IBM	IBM	International Business Machines
International Bu	International Bu	International Business Machines
Little & Sons	Little & Sons	Little and Sons
Little and Son	Little and Son	Little and Sons
MacHenries	MacHenries	McHenrys
MacHenrys	MacHenrys	McHenrys
McHenry's	McHenry's	McHenrys

Applying the New Format

```
title1 "With Format Applied";
proc print data=company noobs;
  var name;
  format name $company.;
run;
```

Listing of Company

Name
International Business Machines
International Business Macnines, Inc.
IBM
Little and Sons
Little & Sons
Little and Son
MacHenrys
McHenrys
MacHenries
McHenry's
Harley Davidson

With Format Applied

Name
International Business Machines
International Business Macnines
International Business Machines
Little and Sons
Little and Sons
Little and Sons
McHenrys
McHenrys
McHenrys
McHenrys
Harley Davidson

Part 3 - Integrity Constraints

- Set of validation rules that can restrict data values from being added, deleted, or updated

- Two general categories
  - General (restrictions within a single file)
  - Referential (involves a reference to a second file)

- Created in PROC DATASETS or PROC SQL
  - We will demonstrate PROC DATASETS

General Integrity Constraints

- CHECK
  Limits data values based on a user-defined constraint.

- NOT NULL
  Disallows missing (null) values.

- UNIQUE
  Requires a specified variable to be unique.

- PRIMARY KEY
  Requires a specified variable or combination of variables to be both nonmissing and unique.

Example Data Set: HEALTH

Listing of Data Set Health

Patno	Gender	HR	SBP	DBP
001	M	88	140	80
002	F	84	120	78
003	M	58	112	.
004	F	66	200	120
007	M	88	148	102
015	F	82	148	88

Goal: Add integrity constraints to this dataset

Creating the Integrity Constraints

```
proc datasets library=work nolist;

modify Health;

   ic create Gender_chk = check
      (where=(Gender in('F','M')))
      message="Gender must be F or M"
      msgtype=user;

   ic create Hr_chk = check
      (where=( HR between 40 and 100))
      message="HR must be between 40 and 100"
      msgtype=user;
```

Limit valid data values for Gender

Limit valid data values for HR

Creating the Integrity Constraints (cont.)

```
   ic create SBP_Chk = check
      (where=(SBP between 50 and 240 or SBP is missing))
      message="SBP must be between 50 and 240 or missing"
      msgtype=user;

   ic create DBP_Chk = check
      (where=(DBP between 35 and 130 or DBP is missing))
      message="DBP must be between 35 and 130 or missing"
      msgtype=user;

   ic create ID_Chk = primary key (Patno)
      message="Patno must be unique and non-missing"
      msgtype=user;

run;
quit;
```

Limit valid data values for SBP

Limit valid data values for DBP

Limit valid data values for Patno

Additional Information in PROC CONTENTS

After Running Integrity Constraints Program

```
ods select IntegrityConstraints;
proc contents data=Health;
run;
```

Alphabetic List of Integrity Constraints

#	Integrity Constraint	Type	Variables	Where Clause
1	DBP_Chk	Check		(DBP>=35 and DBP<=130) or (DBP is null)
2	Gender_Chk	Check		Gender in ('F', 'M')
3	HR_Chk	Check		(HR>=40 and HR<=100)
4	ID_Chk	Primary Key	Patno	
5	SBP_Chk	Check		(SBP>=50 and SBP<=240) or (SBP is null)

Append New Data to the HEALTH Data set

Cells in red are data violations

```
proc append base=Health data=New;
run;
```

Listing of Data Set New

Patno	Gender	HR	SBP	DBP
456	M	66	98	72
567	F	150	130	80
003	M	70	134	86
123	F	66	10	80
013	X	.	120	90

- Violates rule that 40<=HR<=100
- Violates rule that Patno must be unique
- Violates rule that 50<=SBP<=240 or SBP missing
- Violates rule that Gender in (“F” or “M”)

Log Notes Rejecting Observations with Errors

Only One New Observation is Appended

```
NOTE: Appending WORK.NEW to WORK.HEALTH.

WARNING: Patno must be unique and non-missing , 1 observations rejected.

WARNING: Gender must be F or M , 1 observations rejected.

WARNING: SBP must be between 50 and 240 or missing , 1 observations rejected.

WARNING: HR must be between 40 and 100 , 1 observations rejected.

NOTE: There were 5 observations read from the data set WORK.NEW.

NOTE: 1 observations added.

NOTE: The data set WORK.HEALTH has 7 observations and 5 variables.
```

Removing an Integrity Constraint

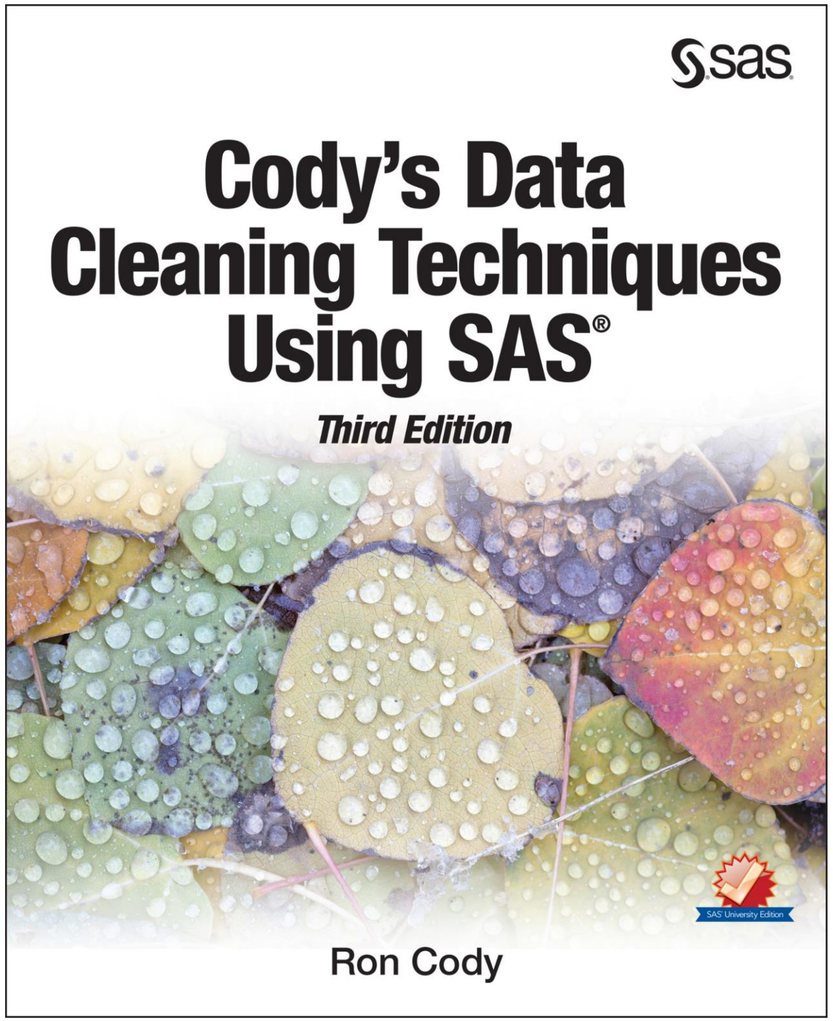
```
proc datasets library=work nolist;
  modify Health;
  ic delete Gender_chk;
quit;

ods select IntegrityConstraints;
proc contents data=Health;
run;
```

#	Integrity Constraint	Type	Variables	Where Clause
1	DBP_Chk	Check		(DBP>=35 and DBP<=130) or (DBP is null)
2	HR_Chk	Check		(HR>=40 and HR<=100)
3	ID_Chk	Primary Key	Patno	
4	SBP_Chk	Check		(SBP>=50 and SBP<=240) or (SBP is null)

Reference

SAS Press Book



- Webinar material comes from this book!
- Available on Redshelf and Amazon.
- Programs and datasets are downloadable for free from the Ron Cody SAS Author Page. Includes several helpful macros!

Thank you!