TransUnion

SAS Efficiencies at TransUnion

With a Focus on PROC FORMAT

MAY 26, 2017

Today's Agenda

Being Efficient by Automating Frequent Processes

- SAS Conversion Code

Being Efficient through PROC FORMAT

- Introduction
- Uses and Examples:
  - Storing descriptions/definitions in a format
  - Binning
  - Efficient Merges

Being Efficient by Automating Frequent Processes

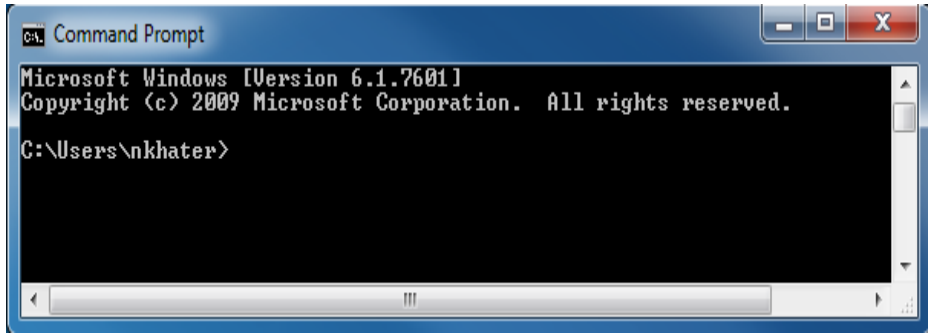
SAS Conversion Code

Being Efficient by Automating Frequent Processes

SAS Conversion Code

- Codes that convert between SAS files and TXT files

- Can even do the conversion from the command prompt (Set up for non-SAS users)



SAS Conversion Code: From TXT to SAS

In the Command Prompt:

Step 1) First, change directory to the folder location of the input file and the expected output file:

```
cd "<folder-location-of-input-and-output-files>"
```

Step 2) Then run the sas code:

```
"<location-of-sas-exe>\sas.exe"
-sysin "<location-of-sas-code>\convert_txttosas.sas"
-set output_filename <name-of-output-file>
-set input_filename <name-of-input-file>
-set delimiter <delimiter-from-input-file>
-set guess_rows <number-of-rows-to-guess-format-from>
```

User Inputs Variables:

- ✓ Folder name of input file
- ✓ Path and name of SAS code
- ✓ Output name
- ✓ Input name
- ✓ Delimiter
- ✓ Guessing Rows

TransUnion

SAS Conversion Code: From TXT to SAS

First, get input macro variables.

```
%let output_filename=%sysget(output_filename); /*output name without extension*/
%let input_filename=%sysget(input_filename); /*input name with extension*/
%let delimiter=%sysget(delimiter); /*delimiter enclosed in quotations*/
%let guess_rows=%sysget(guess_rows); /*aim for 10% of total rows*/
```

Then, decode the delimiters

```
data _null_;
if "&delimiter." = "pipe" then do;
call symput('delimiter',"'|'");
end;

if "&delimiter." = "comma" then do;
call symput('delimiter',"','");
end;

if "&delimiter." = "tab" then do;
call symput('delimiter',"'09'x");
end;

run;

%put &delimiter;
```

and run the import code.

```
libname lib ".";

proc import datafile=".\&input_filename."
     dbms=dlm
     out=lib.&output_filename.
     replace;
   delimiter=&delimiter.;
   guessingrows=&guess_rows.;
run;
```

TransUnion

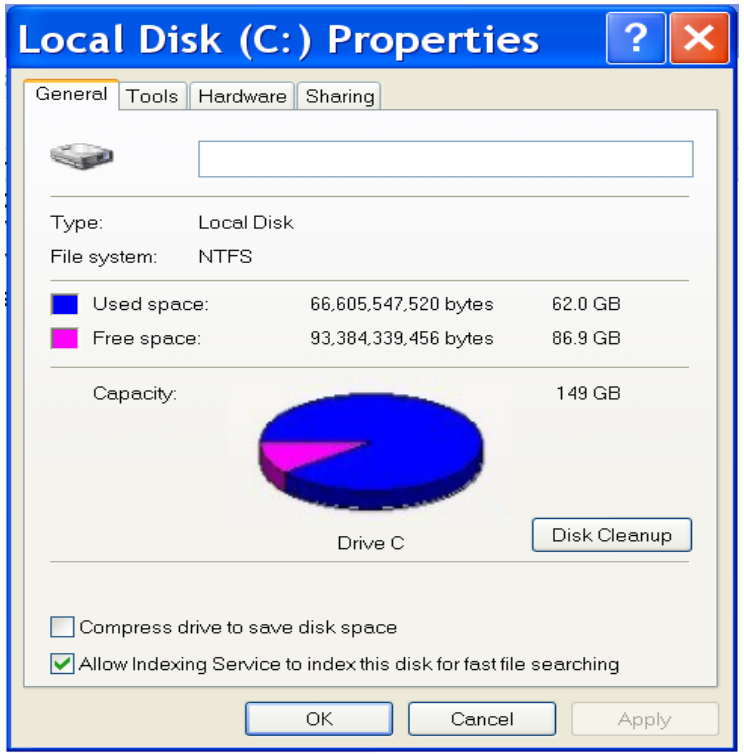
Being Efficient through PROC FORMAT

Being Efficient through PROC FORMAT

Why user-defined formats?

-saves run time

-saves storage space



-simplifies coding

Being Efficient through PROC FORMAT

2 ways to create a user-defined format

- manually assigning labels to values (or ranges of values)
- reading in an existing table that contains values and their corresponding labels (uses the **cntlin** option)

x	y
–3	6
–2	0
–1	–4
0	–6
1	–6
2	–4
3	0

Being Efficient through PROC FORMAT

USES and EXAMPLES:

→ **Storing descriptions/definitions in a format**

- **Look up postal codes to get city, province, population**
- **Look up a person's location by formatting the customer_id**
- **Lender_list table: can make a format that tells you the name of the lender from the lender_code**

Being Efficient through PROC FORMAT

Example 1: Storing descriptions/definitions in a format

First, prepare the cntlin table.

```
data lendercode_to_lendername;
      retain fmtname '$lender_names';
      set lender_list;
      keep lender_code lender_name fmtname;
      rename lender_code=start
             name=label;

 run;
```

Then, read in the table to define the format.

```
proc format cntlin=lendercode_to_lendername;
 run;
```

Now, let's test the format.

```
data test_the_format;
      set test_lendercodes;
      name=put(lender_code, $lender_names.);
 run;
```

	lender_code	lender_name
...	2683	BMO
...	2000	BMO
...	25122	RBC
...	425	TD
...	18	BNS
...	21205	CIBC
...	2691	BMO
...	02519	BNS

TransUnion

Being Efficient through PROC FORMAT

USES and EXAMPLES:

→ Storing descriptions/definitions in a format

- Look up postal codes to get city, province, population
- Look up a person's location by formatting the customer_id
- Lender_list table: can make a format that tells you the name of the lender from the lender_code

→Binning

- **Separating ages/scores into buckets of 5's or 10's (frequently used increments)**
- **Binning each of the TU scores based on standard tiers; no need to lookup ranges every time**

TransUnion

Being Efficient through PROC FORMAT

Example 2: Binning

```
proc format;
     value score1_tiers
              0        = 'Unscoreable'
              .        = 'Unscoreable'
              300-599  = 'Subprime'
              600-699  = 'Near Prime'
              700-779  = 'Prime'
              780-829  = 'Prime Plus'
              830-high = 'Super Prime';
run;
```

```
proc format;
     value score2_tiers
              0        = 'Unscoreable'
              .        = 'Unscoreable'
              300-639  = 'Subprime'
              640-719  = 'Near Prime'
              720-759  = 'Prime'
              760-799  = 'Prime Plus'
              800-high = 'Super Prime';
run;
```

TransUnion

Being Efficient through PROC FORMAT

USES and EXAMPLES:

→Storing descriptions/definitions in a format

- Look up postal codes to get city, province, population
- Look up a person's location by formatting the customer_id
- Lender_list table: can make a format that tells you the name of the lender from the lender_code

→Binning

- Separating ages/scores into buckets of 5's or 10's (frequently used increments)
- Binning each of the TU scores based on standard tiers; no need to lookup ranges every time

→ Making merges more efficient

- **Tagging to subset tables to relevant data**
- **Reading 1 table as a format to be applied to the common variable of the other table (no need to sort)**

Being Efficient through PROC FORMAT

Example 4: Making Merges more Efficient

1. Tag only the relevant from the big table.

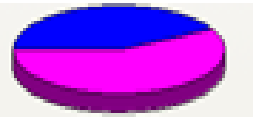
```
data tag_format;
      set smalltable;
      rename customer_id=start;
      label='xxxxxxxxx';
      fmtname='tag';
      keep start label fmtname;
  run;

proc format cntlin=tag_format;
  run;

data bigtable_subset;
      set bigtable;
      where put(customer_id, tag.)='xxxxxxxx';
  run;
```

customer_id	other variables		
xxxxxxxx			
xxxxxxxx			
xxxxxxxx			
xxxxxxxx			

customer_id	other variables		
xxxxxxxx			
xxxxxxxx			
xxxxxxxx			
xxxxxxxx			



Being Efficient through PROC FORMAT

Example 4: Making Merges more Efficient

2a. Perform the merge as usual.

```
proc sort bigtable_subset;
     by customer_id;
run;

proc sort smalltable;
     by customer_id;
run;

data combined;
     merge smalltable (in=a)
           bigtable_subset (in=b);
     by customer_id;
     if a;
run;
```

2b. Apply a format defined from one table to the other table (no need to sort).

```
data score_format;
     set bigtable_subset;
     rename customer_id=start
            score=label;
     fmtname='score';
     keep start label fmtname;
run;

proc format cntlin=score_format;
run;

data smalltable_with_score;
     set smalltable;
     score=put(customer_id, score.);
run;
```

Being Efficient through PROC FORMAT

Things to keep in mind about the PROC FORMAT:

- Naming a format (fmtname):
  - for character formats, $ at beginning
  - no numbers at beginning or end
  - can't be identical to existing SAS format name
- length of fmtname: 32
- Option fmtlib prints the contents of the format
- library= option stores the format permanently

Important fields in a format table:

- ✓ start (required)
- ✓ label (required)
- ✓ fmtname (required)
- ✓ end (for ranges)
- ✓ hlo ('high', 'low', 'other')
- ✓ type (C,N)

Being Efficient through PROC FORMAT

Things to keep in mind about the PROC FORMAT:

The SAS System

FORMAT NAME: SID LENGTH: 2 NUMBER OF VALUES:10000
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 2 FUZZ: STD

START	END	LABEL (VER. V7|V8 21MAY2017:22:33:55)
9077	9077	ON
9078	9078	ON
9085	9085	ON
9086	9086	QC
9087	9087	ON
9090	9090	BC
9092	9092	BC
9093	9093	PE
9094	9094	NB
9095	9095	NS
9096	9096	NL
9097	9097	NS
9099	9099	NS
9100	9100	NB
9101	9101	AB
9102	9102	NS
9103	9103	NS
9104	9104	NB

- Allows for estimated values to show the same label.
- Default: **1e-12**
- Example: fuzz = 0.2 means that label **y** is applied to value **x-0.2** to **x+0.2**

Summary

Being Efficient by Automating Frequent Processes

- SAS Conversion Code
- Automate processes if you can to save coding time and produce consistent results
- Can run SAS from command line, which is useful for simplifying tasks for non-SAS-users

Being Efficient by using PROC FORMAT

- Storing descriptions/definitions in a format
- Binning
- Efficient Merges

Thank you!

Questions? 

References

- Bilenas, Jonas. “I Can Do That With PROC FORMAT” Accessed April 5th 2017. http://www2.sas.com/proceedings/forum2008/174-2008.pdf
- Kupronis, Ben. “PROC FORMAT: An Analyst’s Buddy” Accessed April 5th 2017. http://www2.sas.com/proceedings/sugi31/084-31.pdf
- Shoemaker, Jack. “Ten Things You Should Know About PROC FORMAT” Accessed April 5th 2017. https://stats.idre.ucla.edu/wp-content/uploads/2016/02/bt3014.pdf

Contact Information

Nada Khater
Data Scientist, TransUnion Canada

Work:
nkhater@transunion.com
(905) 320-8697

Personal:
https://www.linkedin.com/in/nadakhater/
khaterna@hotmail.com
(647) 783-0993