PharmaSUG China 2019 - Paper DM-081

Top 10 SAS programming efficiencies

Charu Shankar, SAS® Institute Inc.

ABSTRACT

This practical paper will discuss the Top 10 SAS best programming practices culled from years of experience in working with SAS to help SAS customers resolve their efficiency issues. The reader will be guided on what worked with benchmarking statistics and why a certain practice is a best practice. This session will provide answers to the following questions: “What are 3 questions I need to answer before I jump into working with data”, “What is the data worker’s rule #1?”, “What is the only answer to the question - what’s the best way to do this task?”.

In this paper, participants will learn top 10 SAS best programming practices to improve performance. Participants will learn data access techniques, data manipulation techniques and data output techniques to help conserve valuable resources such as I/O, CPU, Memory and last but not least the programmer’s time. For each best practice the author will demonstrate several ways of performing a task & then, by using benchmarking statistics will show why a certain technique is more efficient. The paper will also compare the data step with the proc step to showcase where the data step has its strength, which proc to use, etc.

DATA USED IN THIS PRESENTATION

TABLE NAME	DETAILS
Cesales_analysis	Contains sales information on 4 categories of chocolate purchased by various customers:gourmet, internet, etc. from 1999-2003.
Ceorder_info	Contains sales information on chocolate orders by cases.
Dec04sales.xls	Contains chocolate sales information in December 2004.

Table 1. Details about the data sets used in this paper

1. INTRODUCTION

The author first wrote this paper in 2011 when she was invited to present at the South-Central SAS Users group in Texas. Since then, this paper has gone through many iterations as more ways of writing code have been revealed through constant enquiry. Every question

served as a curious impetus for the author to research and find more efficient ways to code in SAS. This paper is a humble attempt at capturing and bottling ten of the SAS best practices related to resources such as memory, I/O, storage, CPU time and programmer time. Whether you are a novice user or an experienced SAS coder, the hope is for you to get something new out of this paper. Comments and feedback are always appreciated.

1. What are the 3 Most Important Questions?

Suppose a user came to you for a report. What 3 questions would you ask them before you plunged into data work? I hope you will consider these three vital questions which will prevent any extra alteration and thus avoid extra time which can be very expensive in the long run. Following this sequence of questions will not only aid you in recalling SAS data step sequencing but will also hold you in good stead in other business projects.

1. What do you want – The Output
2. Where is the data – The Input
3. Can I use the data directly or must I shape and transform the data to take me from Input to Output – The Process

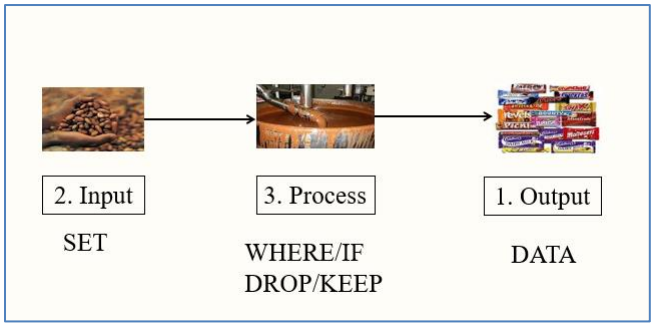


Figure 1. What are the 3 most important questions to ask

What 6 resources are being considered?

Any discussion around computing efficiencies would be effective if we understand what those efficiencies are. The following six resources are being considered in this paper.

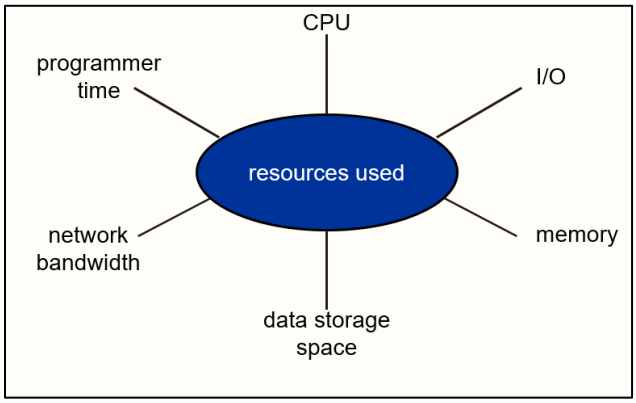


Figure 2. What 6 resources are being considered?

Resource	Definition
CPU	Measurement of time that the central processing unit takes to perform requested tasks such as calcuations, reading and writing data, conditional and iterative logic, and so on.
I/O	Measurement of the read-write operations performed when data and programs are moved from a storage device to memory(input) or from memory to storage or a display device(output)
Memory	Size of the work area required to hold executable program modules, data, and buffers.
Storage	Amount of space that is required to store data on a disk or tape.
Programmer time	Amount of time required for the programmer to write and maintain the program.
Network bandwidth	Amount of data that can pass through a network interface over time. This time can be minimized by performing as much of the subsetting and summarizing as possible on the data host before transferring the results to the loal computer. Network bandwidth is heavily dependent on network loads.

Figure 3. Definition of the 6 Resources being considered

Understanding Efficiency Trade-Offs

A discussion of computing efficiencies will typically lead us to the tradeoff that is also happening in computing. For example, if a user considered space to be an important resource to conserve, then they may use compression techniques(discussed later in this paper). However that could mean an increase in another computing resource, i.e. CPU time as the data needs to be expanded to its full original width in memory in order for SAS to tackle it. Put simply, an increase in one resource can result in a decrease in another. The main understanding of this paper is that its simply not possible to have increase efficiency in all computing resources simultaneously.

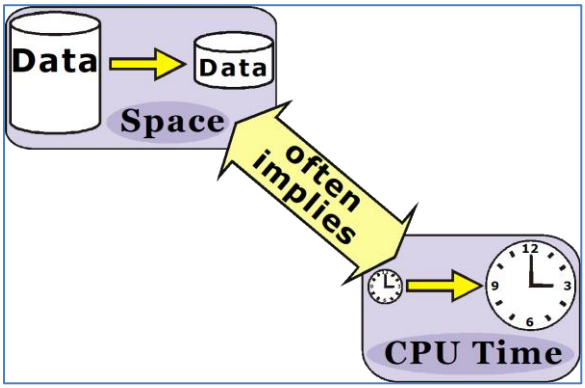


Figure 4. Understanding Efficiency Trade-Offs

2. Saving CPU - What is the data worker's Rule #1?

If life was as easy as an Instagram push on your phone, wouldn't it be a breeze?

However, as data workers and data scientists, its common knowledge that there is no magic pill to swallow that will forgive us for not knowing our data. Knowing our data has

to be the #1 rule for all data workers to look at before plunging into data analysis. Over 80% of a data worker's time is typically spent in gathering knowledge about the data. Cleansing it, scrubbing it, removing incorrect fields and really getting to know the data. time spent on the remaining 20% is typically spent on the cosmetic side of Business intelligence pushing out beautifully formatted reports. Here are two easy ways to get powerful insights on your data using PROC SQL dictionary tables. Can you guess why querying the SASHELP library took so much longer?

Code to query dictionary tables using the SASHELP Library

```
proc print data=sashelp.vcolumn label noobs;
   var libname memname name type length;
         where libname ='CHOC' and upcase(name) contains 'ID';
run;
NOTE: There were 21 observations read from the data set SASHELP.VCOLUMN.
      WHERE (libname='CHOC') and UPCASE(name) contains 'ID';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.58 seconds
      user cpu time       0.03 seconds
      system cpu time     0.04 seconds
      memory              5254.75k
      OS Memory           29548.00k
      Timestamp           07/17/2019 02:11:24 PM
```

Code to query dictionary tables using the Dictionary.columns table

```
proc sql;
   select libname, memname, name, type, length
         from dictionary.columns
               where libname ='CHOC' and upcase(name) contains 'ID';
quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.04 seconds
      user cpu time       0.03 seconds
      system cpu time     0.00 seconds
      memory              5168.93k
      OS Memory           31340.00k
      Timestamp           07/17/2019 02:11:24 PM
```

- When you query a DICTIONARY table, SAS gathers information that is pertinent to that table. Depending on the DICTIONARY table that is being queried, this process can include searching libraries, opening tables, and executing SAS views.
- Unlike other SAS procedures and the DATA step, PROC SQL can improve this process by optimizing the query before the select process is launched. Therefore, although it is possible to access DICTIONARY table information with SAS procedures or the DATA step by using the SASHELP views, it is often more efficient to use PROC SQL instead.

3. CPU Saving – Boiling down data

Hands-down, boiling data must be one of the easiest ways to optimize CPU time in processing. Consider 2 scenarios: subsetting your data at the bottom of the step versus at the top of the step.

Code : Subsetting IF at the bottom of the data step

```
data totals;
      set choc.cesales_analysis;
      do month=1 to 12;
            totcase + total_cases;
            totsales + total_sales;
      end;
      cust='*****';
      custorder=1;
      if customer_type='Gourmet';
run;
```

Code : Subsetting IF as high up as possible

```
data totals;
      set choc.cesales_analysis;
      do month=1 to 12;
            totcase + total_cases;
            totsales + total_sales;
      end;
      if customer_type='Gourmet';
      cust='*****';
      custorder=1;
run;
```

Technique	CPU	I/O	Memory
I. Subsetting IF at Bottom	2.3	1226.0	265.0
II. Subsetting IF near Top	1.3	1226.0	265.0
Percent Difference	42.8	0.0	0.0

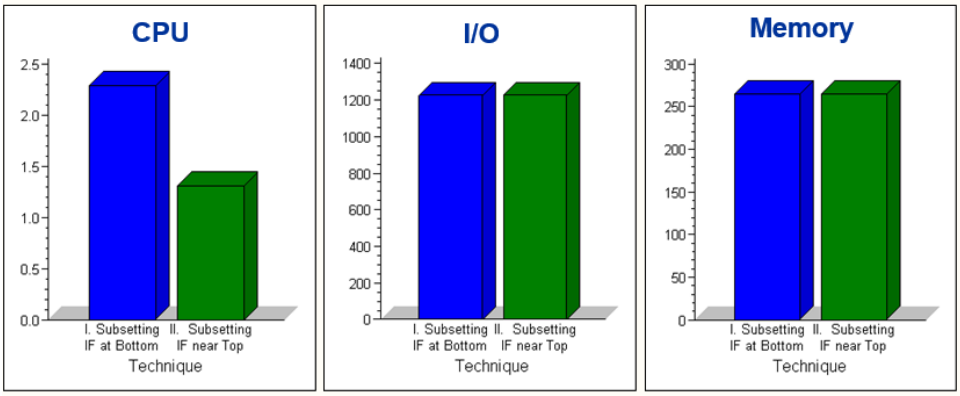
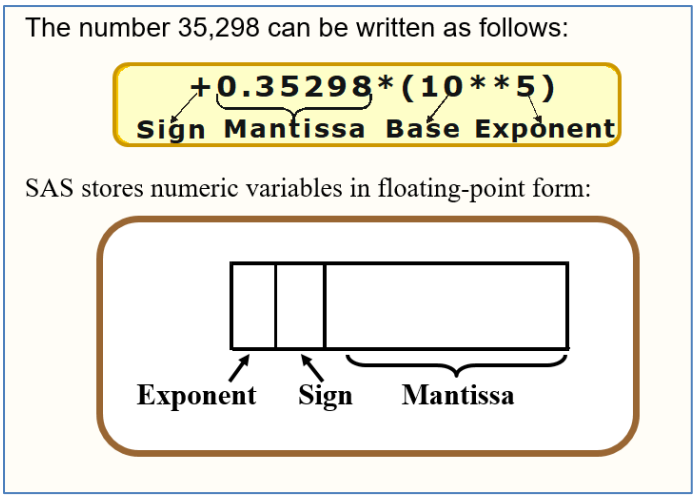


Figure 5. The Result – comparing techniques

4. CPU saving – Do not reduce the length of numeric data

This SAS efficiency tip typically leads into an interesting debate between those who want to reduce the length of numbers to save space and others who are more cautious.

Any reduction in the size of a numeric variable would cause the number to be truncated to the specified length when the value is written to the SAS data set. This reduces the number of bytes available for the mantissa, which reduces the precision of the number that can be accurately stored. It causes the number to be expanded to 8 bytes in the PDV when the data set is read by padding the mantissa with binary zeros. Take a look at the figure below.



. **Figure 6. Default length of Numeric Variables in SAS**

What happens if we try to reduce the length of non-integer data?

Take a look at the code below. Reducing the Length of the variable X to 4 bytes, results in a loss of numeric precision. One tenth in both base 2 and 16 is a number like 1/3 in base 10: It is infinitely repeating. Hence, the difference in the numbers that are stored and read. The second DATA step is necessary, as the truncation is not done until the numbers are written out to a SAS data set.

Code to reduce the length of non-integer data

```
data test;
      length x 4;
      X=1/10;
      Y=1/10;
run;

data _null_;
      set test;
      put X=;
      put Y=;
run;
x=0.0999999642
y=0.1
run;
```

As always, the reader is encouraged to compare pros and cons before weighing in with a final decision.

Advantages	Disadvantages
Conserves data storage space	Uses additional CPU to read
Requires less I/O to read	Can alter high-precision values such as non-integer and large integer values

. **Figure 7. Comparing Pros and Cons of Reduced length numerics**

5. Saving I/O – Reduce multiple passes of your data

If you were watching a favorite TV show, would you get up from the safety of your couch to the kitchen 4 times to get your chocolate fix, or would you rather fill up on all your snack intake at one go so you don't have to make unnecessary trips? Consider 3 techniques below to create 4 subsets from a dataset.

Code to create 4 subsets by using 4 datasteps:

```
data chocolate;
      set choc.cesales_analysis;
      if category='Chocolate' ;
Run;

data gummy;
      set choc.cesales_analysis;
      if category='Gummy' ;
Run;

data hard;
      set choc.cesales_analysis;
      if category='Hard' ;
Run;

data sugarfree;
      set choc.cesales_analysis;
      if category='Sugar-Free' ;
Run;
```

Code to create 4 subsets by using 4 SQL select statements:

```
proc sql;
      create table  chocolate as
            select * from choc.cesales_analysis
                  where category='Chocolate';

create table  gummy as
      select * from choc.cesales_analysis
            where category='Gummy';

create table  hard as
      select * from choc.cesales_analysis
            where category='Hard';
```

```
create table sugarfree as
      select * from choc.cesales_analysis
            where category='Sugar-Free';
quit;
```

Code to create 4 subsets in a single data step:

```
data chocolate gummy hard sugarfree;
      set choc.cesales_analysis;
            if category='Chocolate' then output chocolate;
            else if category ='Gummy ' then output gummy;
            else if category='Hard' then output hard;
            else if category='Sugar-Free' then output sugarfree;
run;
```

I'm sure you would agree that the third technique would be most efficient as its only reading the source data once while the other two techniques need to read the source data 4 times in order to process and create 4 datasets.

6. Saving I/O – Take what you need

Simple techniques can conserve I/O. The amount of I/O saved depends on the size of the subset being processed. Following are 3 ways in which we can take what we need by processing only necessary observations or processing only necessary variables.

6.1 Reduce the number of observations - WHERE in the Data step/PROC step

A subset could be created by using the WHERE statement in a procedure or the data step.

Code to use the WHERE statement in a procedure to create a subset of data:

```
data choc.yearend;
      set phsugxls.'dec04sales$'n;
            extracase=total_cases*2;
run;
proc means data=choc.yearend mean sum;
      where category='Chocolate';
            class customer_type;
                  var extracase;
run;
```

Code to use the WHERE statement in a data step to create a subset of data:

```
data choc.yearend;
      set chocxls.'dec04sales$'n;
            extracase=total_cases*2;
                  where category='Chocolate';
run;
proc means data=choc.yearend mean sum;
      class customer_type;
            var extracase;
run;
```

Did you see how subsetting in the data step is more efficient? It creates a smaller version of the yearend dataset that PROC MEANS has to load into memory for processing.

6.2. WHERE statement or Subsetting IF statement

To create a subset of the cesales_analysis dataset that contains data for Chocolate, either the WHERE clause or the Subsetting IF statement could be used. Let's see how each of these statements plays out in terms of resource usage.

Code to use the Subsetting IF statement in a data step:

```
data chocolate;
      set choc.cesales_analysis;
            if category='Chocolate' ;
Run;

NOTE: There were 115928 observations read from the data set
choc.CESALES_ANALYSIS.
NOTE: The data set WORK.CHOCOLATE has 50368 observations and 11
variables.
NOTE: DATA statement used (Total process time):
```

Code to use the WHERE clause in a data step:

```
data chocolate;
      set choc.cesales_analysis;
            where category='Chocolate' ;
Run;

NOTE: There were 50368 observations read from the data set
choc.CESALES_ANALYSIS.
WHERE category='Chocolate';
NOTE: The data set WORK.CHOCOLATE has 50368 observations and 11
variables.
NOTE: DATA statement used (Total process time):
```

Did you notice the difference in the number of observations read in? Let's learn why the WHERE clause is so efficient in reading in only those observations that are a match.

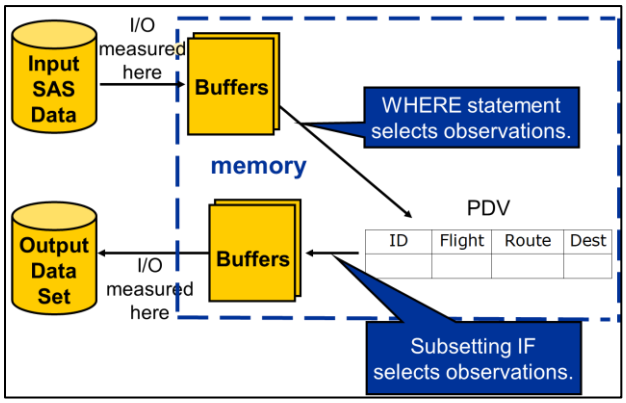


Figure 8. Timing of the WHERE clause vs. the Subsetting IF statement

The reason is simply to do with timing. The powerful WHERE acts on observations before moving them to the Program Data Vector(PDV). The Subsetting IF statement works on newly created variables, but has to read in data, row by row into the PDV thus slower in comparison.

So when in subsetting doubt, simply consider the following efficiency tip.

Use the WHERE clause while subsetting on existing data(or variables coming from the input dataset).

Use the Subsetting IF statement while subsetting new variables (or variables that are being built in the PDV.

6.3 Process only necessary variables

In order to subset variables, you can use the following:
- DROP and KEEP statements
- DROP= and KEEP= data set options

Here is the timing of these two statements and options.

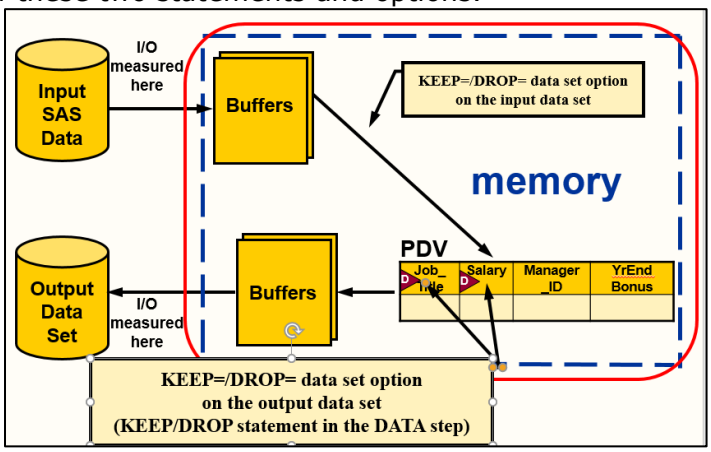


Figure 9. Timing of the DROP/KEEP statements and options

The example below demonstrates one usage of the keep option on the data step.

Code to read all variables and write out 2 variables:

```
data choc.yearend(keep=extracase category);
      set chocxls.'dec04sales$'n;
            extracase=total_cases*2;
                  where category='Chocolate';
run;

proc means data=choc.yearend mean sum;
      class customer_type;
            var extracase;
run;
```

In the following performance comparison. notice that CPU time is consistently the same across techniques. The biggest performance benefit is from I/O Savings.

Technique	CPU	I/O	Memory
I. KEEP not used	2.9	7177	8140
II. KEEP on DATA statement	2.3	656	8138
III. KEEP on SET statement	2.4	1625	8138
IV. KEEP on SET and DATA statements	2.2	662	8138
V. KEEP on SET and PROC statements	2.4	1625	8139

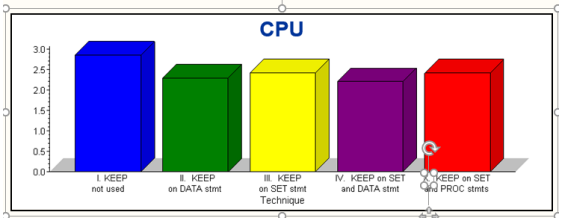


Figure 10. CPU usage comparison using DROP/KEEP statements

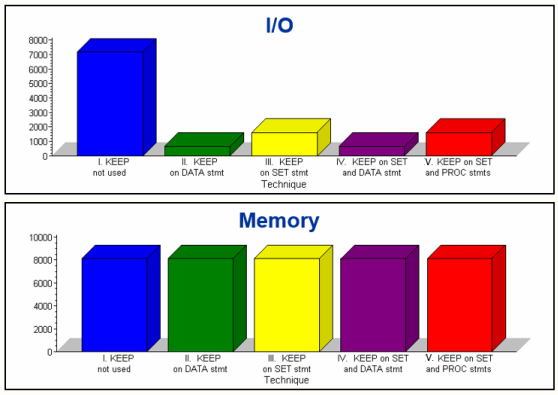


Figure 11. I/O and Memory usage comparison using DROP/KEEP statements

As the above performance graphs indicate, the biggest I/O savings result from reducing the number of variables from the input and the output data set.

7. Saving Space - Store data as character

A definite advantage of storing data as character is space saving. Character data lends itself easily to the Length statement and data does not get truncated by using this function. The following questioning process will illuminate one way of deciding column type.

1. **Space considerations** – How much space does the column use? Which data type-character or numeric is a more efficient space saver?

2. **Data manipulation considerations** – If data is to be extracted out of a column, what data type would be appropriate? The answer clearly is character data which permits the use of the substring function to extract pieces of the column.

3. **Data calculation considerations** – If summary statistics have to be calculated for a column, what data type would be appropriate? The answer, quite obviously is storing the data as numeric which allows for the use of numeric functions such as SUM, MEAN etc. to provide summary statistics for that column.

8. Saving Space – Compression Techniques

SAS data compression can solve both storage space and I/O concerns. Here is a simplified look at an uncompressed and a compressed SAS dataset.

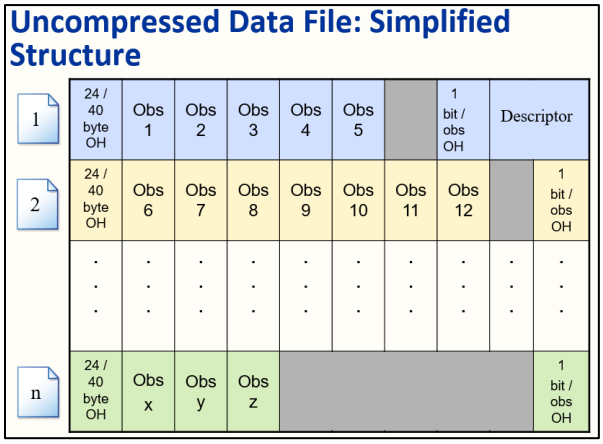


Figure 12. Uncompressed data file: simplified structure

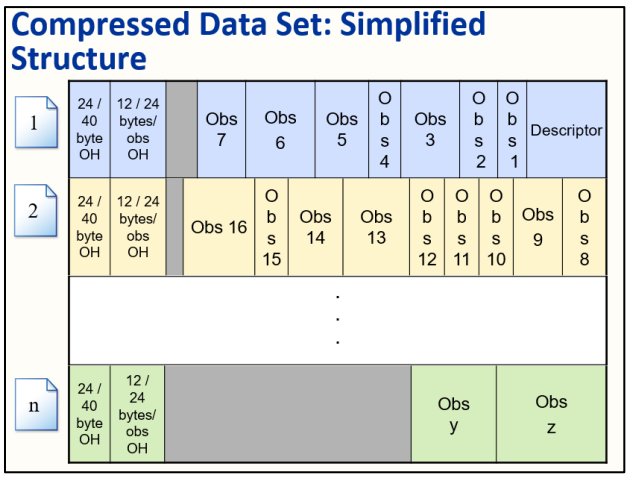


Figure 13. Compressed data file: simplified structure

There are two standard compression algorithms. The optimal algorithm depends on the data. The following chart will explain when to use a certain technique.

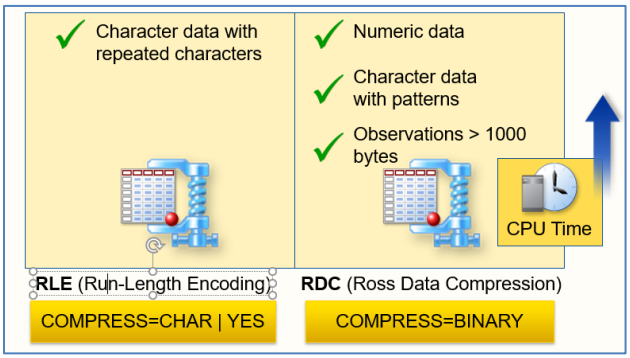


Figure 14. Compression Algorithms

Code to compress a single dataset using the dataset option:

```
SAS-data-set(COMPRESS=NO|YES|CHAR|BINARY);
```

Code to compress all output datasets using a system option:

```
OPTIONS COMPRESS=NO|YES|CHAR|BINARY;
```

Dependencies and Trade-offs in Compression

Compressing SAS data files is ***always*** an experiment. Some SAS data sets generally do compress well if they have many missing values or many observations with few bytes in long character variables.

Log showing that compression decreases the size of the dataset:

```
48   data CustDimComp(compress=binary);
49    set orion.Customerdimmore;
50   run;

NOTE: There were 1500 observations read from the data set
      ORION.CUSTOMERDIMMORE.
NOTE: The data set WORK.CUSTDIMCOMP has 1500 observations and 11 variables.
NOTE: Compressing data set WORK.CUSTDIMCOMP decreased size by 20.00 percent.
      Compressed is 4 pages; un-compressed would require 5 pages.
NOTE: DATA statement used (Total process time):
      real time           0.07 seconds
      cpu time            0.01 seconds
```

Some SAS data files do ***not*** compress well. some data sets don't compress well or at all. Because each observation has higher overhead when compressed, a data file can occupy more space in compressed form than in uncompressed form. This happens when the file has the following characteristics: few repeated characters, small physical size, few missing values, and short text strings.

Log showing that compression would increase the size of the dataset:

```
55   data orders(compress=binary);
56      set orion.orders;
57   run;

NOTE: There were 490 observations read from the data set ORION.ORDERS.
NOTE: The data set WORK.ORDERS has 490 observations and 6 variables.
NOTE: Compressing data set WORK.ORDERS increased size by 100.00 percent.
      Compressed is 2 pages; un-compressed would require 1 pages.
NOTE: DATA statement used (Total process time):
      real time           0.29 seconds
      cpu time            0.12 seconds
```

9. Saving Memory - Use the BY statement instead of the CLASS statement

When you use the CLASS statement and class variables in PROC MEANS, the memory requirements can be substantial. SAS keeps a copy of unique values of each class variable in memory. If PROC MEANS encounters insufficient memory for the summarization of all class variables, you can save memory by using the CLASS and BY statement together to analyze the data by classes. Use BY-group processing instead of CLASS statements TO GROUP DATA in those procedures that support both, especially where you have pre-sorted data or can use an existing index.

BY Statement	CLASS Statement
The data set must be sorted or indexed on the BY variables.	The data set does not need to be sorted or indexed on the CLASS variables.
BY-group processing holds only one BY group in memory at a time.	The CLASS statement accumulates aggregates for all CLASS groups simultaneously in memory.
A percentage for the entire report cannot be calculated with procedures such as the REPORT or TABULATE procedures.	A percentage for the entire report can be calculated with procedures such as the REPORT or TABULATE procedures.

Figure 15. Differences between the BY statement and CLASS statement

10. Saving Programmer Time

Programmer time savings can be accomplished by using strategic coding techniques to streamline code and make it easier to debug and also reduce typing. Two ways to save programmer time are discussed below.

Macros for recent log conversation

Programmers are constantly seeking clever ways to understand the SAS log. A running log can sometimes be overwhelming. Wrapping code using the following macro reduces the amount of time that a programmer may spend trying to examine the log for a single piece of code.

Code for recent log conversation:

```
%put;  %put NOTE:--- %sysfunc(datetime(), datetime19.) ---- ;
data saschoc;
set chocxls.'dec04sales$'n;
run;
%put; %put NOTE:---End of Submitted Code---;
```

The log now shows the code wrapped with the text provided in the macro which also prints out the current date and time.

Log for recent log conversation:

```
---  07AUG2019:15:10:22 ----
45          data saschoc;
46          set chocxls.'dec04sales$'n;
47          run;
NOTE: There were 4640 observations read from the data set
CHOCXLS.'dec04sales$'n.
NOTE: The data set WORK.SASCHOC has 4640 observations and 11
variables.
NOTE: DATA statement used (Total process time):
      real time           0.10 seconds
      cpu time            0.00 seconds


48          %put; %put NOTE:---End of Submitted Code---;

---End of Submitted Code---
```

Variable Lists

Whether you are a seasoned programmer or fresh out of school, I'm sure you will appreciate my last tip which is going to be a typing time saver.

Instead of typing the variables names one by one, use a SAS **variable list** which is an abbreviated method of referring to a list of variable names.

Variable List	Included Variables
x--a	all variables in order of variable definition, from X to A inclusive
x-numeric-a	all numeric variables from X to A inclusive
x-character-a	all character variables from X to A inclusive

Figure 16. Variable list - Name Range Lists

Code for variable lists:

```
* Grab all variables that in alphabetical order from customer_id to month;
proc print data=choc.ceorder_info(obs=100);
     var customer_id--month;
run;

* grab only numeric variables;
proc print data=choc.ceorder_info(obs=100);
     var _numeric_;
run;

* grab only character variables;
proc print data=choc.ceorder_info(obs=100);
     var _char_;
run;
```

REFERENCES

"SAS variable lists". Support.sas.com website.
http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a000695105.htm

Shankar, Charu. "Know Thy Data: Techniques for Data Exploration". Pharmasug 2018, Seattle, WA.
https://www.pharmasug.org/proceedings/2018/BB/PharmaSUG-2018-BB11.pdf

"Techniques for Optimizing Memory Usage". Support.sas.com website.
https://go.documentation.sas.com/?docsetId=lrcon&docsetTarget=n1aghi7e4yv3e0n19sn1ho5mmg8e.htm&docsetVersion=9.4&locale=en

"Compress= Data set Option". Support.sas.com website.
http://support.sas.com/documentation/cdl/en/ledsoptsref/69751/HTML/default/viewer.htm#n014hy7167t2asn1j7qo99qv16wa.htm

- Lafler, Kirk. "Exploring DICTIONARY Tables and Views". SAS Users Group International 2005, Philadelphia, PA
http://www2.sas.com/proceedings/sugi30/070-30.pdf

- Homes, Michelle. "Read what you need" . Blogs.sas.com.
https://blogs.sas.com/content/sastraining/2013/08/06/read-what-you-need/

Group processing CLASS vs BY
https://newonlinecourses.science.psu.edu/stat480/node/95/

ACKNOWLEDGMENTS

The author is grateful to the many SAS users that have entered her life. Each User has either asked or answered a question, which in turn gave the author the impetus to research and study more efficient ways of performing SAS tasks. The users are too many to thank individually so this is a thank you to every single user who has touched her life. Charu is grateful to the Pharmasug committee for inviting her to present this paper. She would also like to express her gratitude to her manager, Stephen Keelan without whose support and permission, this paper would not be possible.

CONTACT INFORMATION

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

Charu Shankar
SAS Institute Canada, Inc.
Charu.shankar@sas.com
https://blogs.sas.com/content/author/charushankar/