PhUSE EU Connect 2018

Paper CT10

Big Data Sets Seen as a Big Problem and How to Deal with Them

Daniil Shliakhov, Intego Group, Kharkiv, Ukraine

ABSTRACT

Sometimes SAS programmers experience difficulties while using data sets with a big number of observations. The main problem here is that running SAS programs takes ages if you are working with really huge data sets. Hash objects can partially solve the problem, but the question is how many programmers use hash objects in their daily routine? The paper gives an introduction to some basic techniques that can be helpful in the process of manipulating big data quickly and efficiently, calculating frequencies and descriptive statistics, which could be useful for any SAS programmer. To prove the effectiveness of various methods, the paper also provides comparison between them. It shows how much time is consumed as a resource when using large SAS data sets. Standard SAS procedures such as PROC MEANS, PROC SUMMARY, PROC SQL, PROC FREQ have also been used in the paper.

INTRODUCTION

Working with data sets that contain a normal number of observations is not a problem. But what about data sets with one or even ten million observations? The main problem is running time. When you work with large data sets, it may take minutes or even hours to run SAS programs. Let's imagine a situation when a summary table for pooling study needs to be created. When creating this table, SAS program can contain some DATA or PROC steps to sort data sets, SAS procedures like PROC MEANS to calculate descriptive statistics (mean, median, standard deviation, etc.) or PROC FREQ to calculate the number of patients and percentages. In cases where a data set with a few million observations is used, it really may take a lot of time to run the program. This paper describes different techniques and tips that can be helpful when working with big data sets.

At the end of this paper, comparison of the run time of different techniques will be shown. All examples provided in this paper are based on an ADLB data set with more than 10 million observations that has been created for a pooling study. The structure of the data set is the following: 1 record per study (STUDYID), patient (USUBJID), laboratory parameter (PARAM, PARAMCD), visit (AVISIT, AVISITN). Each patient has been assigned to one of a few treatment groups (TRT01A, TRT01AN).

To run the SAS codes provided in this paper, SAS 9.4 for Microsoft Windows has been used.

All SAS procedures below (like PROC MEANS, PROC SUMMARY, etc.) will be run using SAS data set ADLB to show the difference in running time. It is predefined that data set is sorted before passing to any SAS procedures covered in this paper. Sorting issue will not be covered in this paper at all.

GENERAL TIPS. RETRIEVING SAS DATA SETS

USE VIEW OPTION TO READ SAS DATA SETS

Let's look at the following example.

```
data adlb;
   set adam.adlb;
run;
```

This is a very simple data step and there is a question: what is strange about it? Let's look at the running time of this step when the data set has more than 10 million observations:

Real time	34:23
CPU time	3:10

There is a way to save some running time. The same data step could be run with the VIEW option added. If we look at the definition of SAS Views, it says the following: A SAS view is a type of SAS data set that retrieves data values from other files. A SAS view contains only descriptor information such as the data types and lengths of the variables (columns), plus information that is required for retrieving data values from other SAS data sets.
So, it's time to run the previous DATA step but with the VIEW option added.

```
data adlb / VIEW = adlb;
   set adam.adlb;
run;
```

Now it's time to look at the result:

Real time	0:03
CPU time	0:01

So, as you can see, running a simple data step with the VIEW option will save both time and storage space.
To produce summary table, ADLB data set should be merged with a subject-level data set called ADSL and then the result data should be sorted by a number of variables. Let's look at example below:

```
data analysis;
   merge adam.adlb adam.adsl;
   by studyid usubjid;
run;

proc sort data=analysis;
   by trt01an parcat paramcd avisitn;
run;
```

Let's see how much time it took to run these data step and PROC step:

	Data Step	PROC step
Real time	36:04	1:12.40
CPU time	8:18	12.45

Let's look at the same steps but with using VIEW option:

```
data analysis / VIEW = analysis;
   merge adam.adlb adam.adsl;
   by studyid usubjid;
run;

proc sort data=analysis;
   by trt01an parcat paramcd avisitn;
run;
```

	Data Step	PROC step
Real time	0:04	1:17.65
CPU time	0:02	20.32

So, as shown in the result above, VIEW option can save running time for sure.

USE WHERE OR IF STATEMENTS TO SUBSET DATA

There are two ways to subset data. Let's look at two different DATA steps below:

```
data adlb;
   set adam.adlb;
   if ANL01FL = "Y";
run;

data adlb;
   set adam.adlb;
   where ANL01FL = "Y";
run;
```

The result will be the same after running each of these DATA steps, but if we look at the difference between running times, we can be see that the DATA step with the WHERE statement works quicker:

	IF statement	**WHERE statement**
Real time	31:64	33:31
CPU time	3:53	5:68

What is the difference between IF and WHERE statements? The main difference is that the IF statement subsets data after processing data, while the WHERE statement subsets the data set before processing. Based on the comparison above, IF works quicker than WHERE statement. Even if to use any complex subset conditions, IF will work quicker.
This is quite unpredictable result. But if to run the same data steps and use data set ADLB with index defined (out of scope of this paper), WHERE will be quicker for sure. Look at the table with the comparison below:

	IF statement	**WHERE statement**
Real time	32:15	27:26
CPU time	4:28	2:98

DESCRIPTIVE STATS

Once needed data for the summary table has been retrieved, it's time to calculate some statistics. There are different ways to calculate descriptive statistics like mean, median, standard deviation. Sometimes people may use PROC MEANS, PROC SUMMARY, PROC UNVIARIATE or even PROC SQL to calculate those stats. Usually these SAS procedures provide the same result, but SAS programmers can use one or another based on their preferences. Let's look at all of these procedures and compare how much time it may take to run using each of these procedures.

PROC MEANS

PROC MEANS is the most popular SAS procedure that can calculate standard descriptive statistics in a simple way. The following statement can be found in the SAS Procedure Manual: "The MEANS procedure provides data summarization tools to compute descriptive statistics across all observations and within groups of observations."
Let's calculate mean, median, minimum, maximum and standard deviation using PROC MEANS:

```
proc means data=adlb noprint;
   by trt01an parcat paramcd avisitn;
   var aval;
   output out=mnout n=n mean=mean median=median std=std min=min max=max;
run;
```

This piece of code will calculate predefined descriptive stats based on the requirements.
Let's see how much time it took to run this PROC step:

Real time	15:14
CPU time	3:38

PROC UNIVARIATE

The SAS Procedure Manual says: "The UNIVARIATE procedure provides a variety of descriptive measures, high-resolution graphical displays, and statistical methods, which you can use to summarize, visualize, analyze, and model the statistical distributions of numeric variables. These tools are appropriate for a broad range of tasks and applications". So, based on this, PROC UNIVARIATE can be used to calculate the same statistics as PROC MEANS.
The following PROC step will provide the same result as the previous MEANS procedure:

```
proc univariate data=adlb noprint;
   by trt01an parcat paramcd avisitn;
   var aval;
   output out=mnout n=n mean=mean median=median std=std min=min max=max;
run;
```

Now it's time to check the result and see if PROC UNIVARIATE works quicker than PROC MEANS:

Real time	24:78
CPU time	1:76

PROC SUMMARY

If PROC MEANS is one of SAS's original procedures and its goal was to create printed tables of summary statistics, PROC SUMMARY was introduced to create summary data sets.
A simple run of the SUMMARY procedure can be found below:

```
proc summary data=adlb noprint;
   by trt01an parcat paramcd avisitn;
   var aval;
   output out=mnout n=n mean=mean median=median std=std min=min max=max;
run;
```

The time spent to run this PROC step can be found below:

Real time	13:24
CPU time	3:33

PROC SQL

The SQL procedure is one of the wonderful tools for summarizing data. It provides a number of useful summary functions to calculate descriptive statistics:

```
proc sql noprint;
   create table mnout as
   select trt01an, parcat, paramcd, avisitn, count(*), MEAN(aval) as mean,
MEDIAN(aval) as median, STD(aval) as std, MIN(aval) as min, MAX(aval) as max
   from adlb
   group by trt01an, parcat, paramcd, avisitn;
quit;
```

Please note here that if you are using SAS 9.2, PROC SQL cannot calculate median, because median is a row-wise not and an aggregate function in the SQL procedure. However, beginning with SAS 9.4, MEDIAN is also an aggregate function and can be used with PROC SQL. Let's see how much time was spent to calculate descriptive stats:

Real time	13:45
CPU time	2:53

FREQUENCY

Like for descriptive stats, SAS has a lot of different ways to calculate the number of patients and percentages. The main idea is to create a summary table with number of patients with non-missing laboratory result per each treatment group, laboratory parameter and visit. It's predefined that data set was filtered already before passing to any SAS procedures listed below. Let's look at different SAS procedures that can help to calculate the number of patients and see what procedure is quicker.

PROC FREQ

The FREQ procedure is one of the most popular and main SAS procedures to calculate frequency.

Let's look at an example below and calculate the number of patients with lab results that are outside of normal limits:

```
proc freq data=adlb noprint;
   by trt01an parcat paramcd;
   tables avisitn / out=frout;
run;
```

Let's see how much time was spent to run this piece of code:

Real time	13:62
CPU time	2:04

PROC SQL

Like for descriptive stats, the SQL procedure can be used to calculate the number of patients:

```
proc sql noprint;
   create table frout as
   select trt01an, parcat, paramcd, avisitn, count(*) as count
   from adlb
   group by trt01an, parcat, paramcd, avisitn;
quit;
```

The result of running can be found below:

Real time	12:19
CPU time	1:63

PROC SUMMARY

Finally, the last but not the least is the SUMMARY procedure. Let's see how to use this SAS procedure to calculate the number of patients and the result.

```
proc summary data=adlb nway noprint;
   by trt01an parcat paramcd avisitn;
   output out=frout;
run;
```

The result of running can be found below:

Real time	12:02
CPU time	0:88

CONCLUSION

SAS has a lot of ways to retrieve data and a number of procedures that can calculate some "standard" statistics. It's fine to use any of them when you are working with data sets with a small number of observations. In cases when a data set is large it would be great to think about the right SAS procedure. Let's look at the final comparison of different SAS procedures that can calculate a standard set of descriptive statistics:

	PROC MEANS	**PROC UNIVARIATE**	**PROC SUMMARY**	**PROC SQL**
Actual time	15:14	24:78	**13:24**	13:45
CPU time	3:38	1:76	**3:33**	2:53

Based on this comparison, it's obvious that PROC SUMMARY is the quickest SAS procedure to calculate descriptive stats.

Let's also look at the final comparison of different SAS procedures that can calculate a number of patients (frequency):

	PROC FREQ	**PROC SQL**	**PROC SUMMARY**
Actual time	13:62	12:19	**12:02**
CPU time	2:04	1:63	**0:88**

Based on comparison above, PROC SUMMARY is the quickest SAS procedure to calculate number of patients as for descriptive stats.

CONTACT INFORMATION

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

Daniil Shliakhov
Intego Group
43/2 Gagarin Ave., Kharkiv, Ukraine
daniil.shlyakhov@intego-group.com
https://intego-group.com