Adding One Value to All Observations
Anne W. Warren, MDRC, New York, NY

ABSTRACT
This paper reviews four techniques for adding one value, such as a summary statistic, to all observations in a SAS® data set: "if _N_ = 1 then set datasetname"; PROC SQL; PROC SQL INTO a macro variable; and CALL SYMPUTX to create a macro variable. Programmers can use these methods to avoid hard coding and to write more flexible code. Pros and cons of each technique are reviewed. This paper also discusses situations in which one method may be more useful than others. Real-life examples are provided for each technique. This paper requires a basic understanding of the DATA step, PROC SQL, and macro variables.

INTRODUCTION
Adding one value, such as a summary statistic, to all observations in a data set is easy to do if one is content with hard coding. For example:

```sas
data new;
set old;
x=1;
run;
```

If one desires the code to be as flexible and dynamic as possible, however, then alternate strategies must be employed.

This paper discusses four techniques for automatically adding one value to all observations. Two methods use PROC SQL and two methods use the DATA step. Questions to consider when deciding which method to use include:

1. Can the value be calculated using PROC SQL?
2. Will the value be used in subsequent steps?

Each of the four techniques includes an example and a discussion of its main advantages and disadvantages.

SAMPLE DATA SET
A temporary SAS data set, TEMP, is used in the following four examples. This data set has two variables and ten observations. The first variable is a unique identifier (ID) and the second variable indicates hourly wage (EARNHR). Note that the first record has a missing value for hourly wage.

Print of data set TEMP

<table>
<thead>
<tr>
<th>Obs</th>
<th>ID</th>
<th>EARNHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5.75</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5.75</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6.30</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6.75</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8.00</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8.25</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9.60</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>10.05</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>12.80</td>
</tr>
</tbody>
</table>
ADDING ONE VALUE WITH PROC SQL

Suppose you would like to add a summary statistic for observations in a data set to all observations in the same data set. Using SQL’s aggregate functions to calculate a summary statistic and to add it to all observations is one solution.

EXAMPLE
If you would like to add a mean to all observations (e.g., to calculate z-scores), then the first step might be to calculate the average hourly wage for all observations. Next, this value would be added to all observations in the data set.

CODE
PROC SQL can do this in a single step. The query below tells SAS to create a data set named TEMP1 from the data set TEMP. This new data set will include all (*) the variables and observations from the original data set as well as a new variable, AVGEARNHR. For each record in the data set, AVGEARNHR will have a value that is equal to the average of the variable, EARNHR. Note that the format is not necessary but makes the output easier to read.

```
proc sql;
create table temp1 as
select *
  , mean(EARNHR) as AVGEARNHR format = 8.2
from temp;
quit;
```

OUTPUT DATA SET
The output data set looks as follows.

<table>
<thead>
<tr>
<th>Obs</th>
<th>ID</th>
<th>EARNHR</th>
<th>AVGEARNHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.</td>
<td>8.14</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5.75</td>
<td>8.14</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5.75</td>
<td>8.14</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6.30</td>
<td>8.14</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6.75</td>
<td>8.14</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8.00</td>
<td>8.14</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8.25</td>
<td>8.14</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>9.60</td>
<td>8.14</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>10.05</td>
<td>8.14</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>12.80</td>
<td>8.14</td>
</tr>
</tbody>
</table>

Note that all observations in the data set have the same value for AVGEARNHR, even when EARNHR has a missing value.

PROS AND CONS
The main advantages of using PROC SQL are that it is easy to tell how the summary statistic was calculated and it only requires a few lines of code. One constraint is that there are a limited number of aggregate functions available in SQL that can be used to calculate summary statistics.

ADDING ONE VALUE WITH SQL INTO :MACRO VARIABLE
In situations where the value added to all observations can be calculated with SQL and the value will be used in multiple, subsequent steps, then using SQL INTO to create a macro variable can be helpful.
EXAMPLE
Suppose you have hourly earnings data and would like to create a data set consisting only of observations with earnings that fall below the average. First, PROC SQL can be used to create a macro variable equal to the average hourly wage. This macro variable then can be used in subsequent steps to keep only records where hourly wage falls below the mean.

CODE
The PROC SQL technique shown below is very similar to the previous method except it uses an "into" clause to create a macro variable that has a value equal to the mean, non-missing hourly wage value. Please note that "into" must be followed with :name in order to create a macro variable. Next, the macro variable, &MEANEARNHR, is used in a DATA step to keep only select records.

```
proc sql;
select
  mean(EARNHR) as AVGEARNHR format = 8.2 into :MEANEARNHR
from temp;
quit;

data temp2;
set temp;
  where EARNHR ge 0 and EARNHR lt &MEANEARNHR;
run;
```

OUTPUT DATA SET
The output data set looks as follows. As in the previous example, the mean of EARNHR is $8.14. Therefore, only observations where EARNHR is less than $8.14 (and non-missing) are kept in the data set TEMP2.

```
Obs  ID  EARNHR
 1   2   5.75
 2   3   5.75
 3   4   6.30
 4   5   6.75
 5   6   8.00
```

PROS AND CONS
In addition to the previously noted advantages and disadvantages of using PROC SQL to add one value to all observations in a data set, PROC SQL combined with a macro variable allows one to use the value in subsequent steps.

ADDING ONE VALUE WITH A DATA STEP
Perhaps the most common of the four methods for adding one value to all observations in a data set is by "setting" a data set using "if _N_ = 1 then set datasetname". This technique is useful when the added value cannot be calculated with SQL.

EXAMPLE
Suppose you would like to add the median wage to each record in your data set. SQL cannot calculate the median, so a different strategy must be employed.

CODE
This technique uses the automatic variable _N_ to "merge" a one-record data set to a multiple-record data set. In the PROC MEANS step, the median value for EARNHR is calculated and saved in a data set called ADD. Note that the variables _TYPE_ and _FREQ_ are dropped because they are not relevant to the discussion.
```sas
proc means data = temp noprint;
var EARNHR;
  output out = add (drop = _TYPE_ _FREQ_) median(EARNHR) = MEDIANEARNHR;
run;
```

The resulting data set looks as follows.

**Print of data set ADD**

<table>
<thead>
<tr>
<th>Obs</th>
<th>MEDIANEARNHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.00</td>
</tr>
</tbody>
</table>

In the next step, two data sets are SET together. Data set ADD is SET using "if _N_ = 1 then set **datasetname**", which allows the single record from this data set to be added to all observations in the original data set, TEMP.

```sas
data temp3;
  if _N_ = 1 then set add;
  set temp;
run;
```

This method is not necessarily intuitive unless one understands the way SAS creates a data set. As SAS creates a new data set, it reads one record at a time, saving values from that record in the Program Data Vector (PDV) until values from the next record replace them. SAS continues in this way until it has reached the last record.

For example, when creating the data set TEMP3, the values from the variables MEDIANEARNHR, ID, and EARNHR are added to in the PDV (see the left-hand box, below). When the second record is created, values associated with the second record overwrite the first record’s values (see the right-hand box, below). If a variable is not encountered in the second record, but it is still in the PDV from the first record (i.e., MEDIANEARNHR), then the value for this variable from the first record is still kept in the PDV for the second record. Thus, the MEDIANEARNHR value is added to each record in the data set TEMP3.

<table>
<thead>
<tr>
<th>PDV of TEMP3: 1st record</th>
<th>PDV of TEMP3: 2nd record</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIANEARNHR 8.00</td>
<td>MEDIANEARNHR 8.00</td>
</tr>
<tr>
<td>ID 1</td>
<td>ID 2</td>
</tr>
<tr>
<td>EARNHR .</td>
<td>EARNHR 5.75</td>
</tr>
</tbody>
</table>

**OUTPUT DATA SET**

The resulting data set looks as follows.

**Print of data set TEMP3**

<table>
<thead>
<tr>
<th>Obs</th>
<th>MEDIANEARNHR</th>
<th>ID</th>
<th>EARNHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.00</td>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>8.00</td>
<td>2</td>
<td>5.75</td>
</tr>
<tr>
<td>3</td>
<td>8.00</td>
<td>3</td>
<td>5.75</td>
</tr>
<tr>
<td>4</td>
<td>8.00</td>
<td>4</td>
<td>6.30</td>
</tr>
<tr>
<td>5</td>
<td>8.00</td>
<td>5</td>
<td>6.75</td>
</tr>
<tr>
<td>6</td>
<td>8.00</td>
<td>6</td>
<td>8.00</td>
</tr>
<tr>
<td>7</td>
<td>8.00</td>
<td>7</td>
<td>8.25</td>
</tr>
<tr>
<td>8</td>
<td>8.00</td>
<td>8</td>
<td>9.60</td>
</tr>
<tr>
<td>9</td>
<td>8.00</td>
<td>9</td>
<td>10.05</td>
</tr>
<tr>
<td>10</td>
<td>8.00</td>
<td>10</td>
<td>12.80</td>
</tr>
</tbody>
</table>
PROS AND CONS
There are many advantages to using a DATA step technique: it is easy to use; all programming occurs within a
DATA step; and it is possible to add a value from outside the data set to which it is applied. As mentioned previ-
ously, however, this method is not very intuitive.

ADDING ONE VALUE WITH CALL SYMPUTX
CALL SYMPUTX\(^1\) is useful for adding a value to all observations in your data set when this value cannot be cal-
culated using SQL and you would like to be able to use the value in multiple, subsequent steps.

EXAMPLE
Suppose you would like to create a data set consisting only of observations with wage values that fall below the
median. As mentioned earlier, SQL cannot calculate the median value. Instead, CALL SYMPUTX is used to cre-
ate a macro variable that is equal to the median wage.

CODE
The code below first uses PROC MEANS to create data set ADD2 that contains one variable, MEDIANEARNHR,
which is the median hourly wage in the data set TEMP. So far, this is similar to the previous example.

Next, MEDIANEARNHR is assigned to a macro variable MDNEARNHR\(^2\) using CALL SYMPUTX in a DATA step.\(^3\)

Finally, this macro variable is used to create a data set that only has observations where hourly wage values are
less than the median.

    proc means data = temp noprint ;
    var EARNHR ;
    output out = add2 (drop=_TYPE_ _FREQ_) median(EARNHR) = MEDIANEARNHR;
    run ;

    data _null_ ;
    set add2 ;
    call symputx ('MDNEARNHR', MEDIANEARNHR) ;
    run ;

    data temp4 ;
    set temp ;
    where EARNHR ge 0 and EARNHR lt &MDNEARNHR ;
    run ;

OUTPUT DATA SET
The print below shows the data set TEMP4. (As we saw in the previous example, the median value for EARNHR
is $8.00.)

    Print of data set TEMP4

    Obs    ID    EARNHR
    1      2     5.75
    2      3     5.75
    3      4     6.304
    4      5     6.75

\(^1\) CALL SYMPUTX is a new feature of SAS 9. It is similar to CALL SYMPUT except it trims leading and trailing
blanks.

\(^2\) Note that the name of the macro variable created using CALL SYMPUTX must be in quotes.

\(^3\) The macro variable generated with CALL SYMPUTX is not available for use until after the step in which it is cre-
ated.
PROS AND CONS
The main advantages of using CALL SYMPUTX to create a macro variable is that it is very flexible, so the full power of the DATA step is available. There are no limitations to the value you calculate, and the value can be used for the rest of the program. The main disadvantage is that the macro variable is not available for use until the step after the one in which it is created. Additionally, it is not a very intuitive method.

CONCLUSIONS
All four strategies for adding one value to all observations in a data set work well. Some are easier and more intuitive than others. If the value can be calculated using SQL, then this is a quick and easy method to use. If the value will be used in multiple steps, then create a macro variable using SQL or a DATA step.

REFERENCES
SAS 9.1 SQL Procedure,

ACKNOWLEDGMENTS
SAS® and all other SAS® Institute Inc. product or service names are registered trademarks or trademarks of SAS® Institute Inc. in the USA and other countries. ® indicates USA registration.

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

Anne W. Warren
MDRC
16 East 34th Street, 19th floor
New York, NY 10016
Work Phone: 212.340.8661
Fax: 212.684.0832
Email: anne.warren@mdrc.org
Web: www.mdrc.org