

Analytical Software: SAS

Introduction

SAS is a power statistical analysis tool that is used in many different industries and by many companies. SAS enables you to organize and analyze data using both graphical and analytical methods. The software is flexible, dynamic, and is frequently updated to include statistical and econometric features that are emerging in the professional fields.

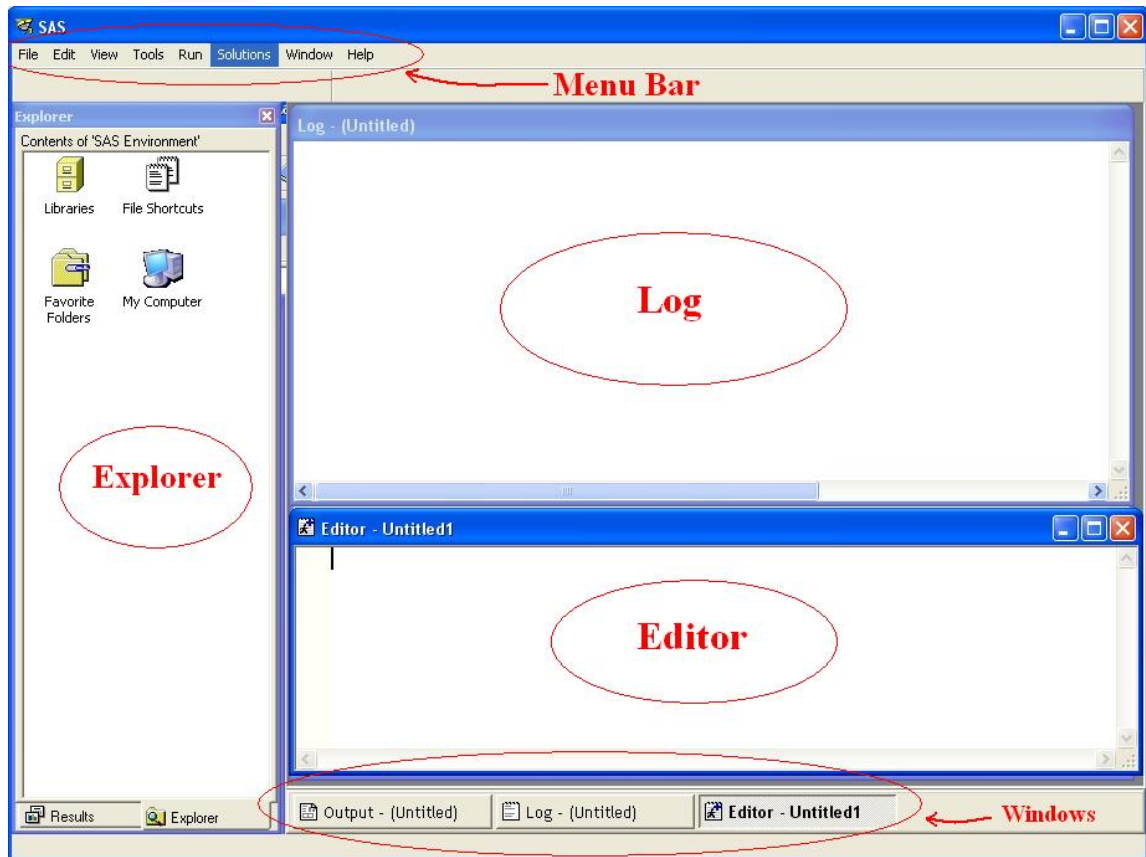
Starting SAS (Windows OS)

Use the Windows **Start** menu to navigate to the SAS executable as follows:

Start → All Programs → SAS → SAS 9.2 (English)

Initial View

When you launch SAS, you will see the following initial view:



- **Log:** Used to examine system messages. The log shows errors in and successful command execution.
- **Editor:** Used to enter programming commands, which tell SAS how the user wants to analyze the data.
- **Explorer:** Used to navigate to active data sets and results. Click the appropriate tab at the bottom of the Explorer panel.
- **Output (*on Windows bar*):** Used to view the analytical results.
- **Menu Bar:** Used for purposes such as opening and saving files, opening various panels, and setting preferences.

NOTE: If you close any of the panels that are open in the initial view, you can do the following to re-open them:

1. Click **View** in the Menu Bar.
2. Select the appropriate panel:

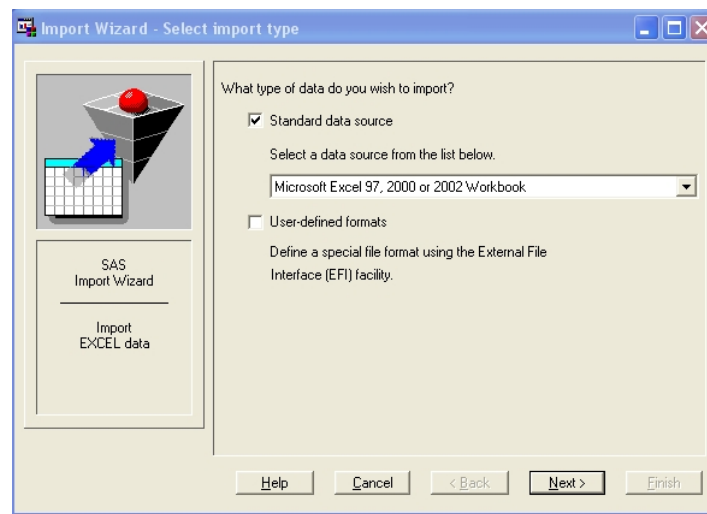
Log	→	Opens the log panel
Enhanced Editor	→	Opens/focuses on the command editor
Output	→	Opens the output panel
Contents Only	→	Opens the Explorer panel

Importing Data

To import data, you can use the SAS Import Wizard as follows:

1. **IMPORTANT:** If you opened the data file in Excel, you need to close Excel before beginning the Import Wizard. Otherwise, you will receive a SAS error and the data will not be imported.
2. In SAS, click **File** in the Menu Bar and select **Import Data**. The Import Wizard window opens.
3. In the “Select a data source from the list below” field, select **Microsoft Excel 97, 2000 or 2002 Workbook** (this is the default). Then, click **Next**. The Connect to MS Excel window opens.
4. In the Connect to MS Excel window, click **Browse**. Navigate to the data file, select it, and then click **OK**.
5. In the “What table do you want to import” field, select the appropriate sheet. Then, click **Next**. **NOTE:** The data files that I provide will only have a single sheet.

6. In the “Library” field, select **WORK** (default).
7. In the “Member” field, enter a name for the data set. Make it something intuitive. For example, if analyzing a data set of wheat prices, name the data set *wheat_data*.
8. Click **Finish** to complete the Import Wizard. The Import Wizard window closes.




Checking Data Import

There are two ways to check whether the data was successfully imported and can be used for statistical analyses. These are as follows:

1. Check the Log panel. A successful import will result in the following log message:
NOTE: WORK.WHEAT_DATA was successfully created.
2. Check the Explorer panel.
 - (a) Click the **Explorer** tab at the bottom of the Explorer panel.
 - (b) Double-click the **Libraries** icon.
 - (c) Double-click the **Work** icon.



You should now see a data table icon  that is named with the designated data name. In this example, the icon would be named **Wheat_data**.

Coding in SAS

To successfully write code within SAS, you can follow these guidelines:

- All code is written in the Editor panel.
- The code is often color coded:


BLUE → Syntactically correct
RED → Incorrectly coded

- **Every** line of code must end with a semicolon (;) (Most errors will stem from users forgetting to end a line with a semicolon).
- Specify the data set that you are using by including the `data = your_dataset` statement following any `proc ...` statement. For example,

```
proc means data=wheat_data;
```

- After finishing writing a snippet of code, you must end the snippet with the `run;` statement. For example,

```
proc means data=wheat_data;  
run;
```

- To execute the written code, click the **Run** icon  found in the Icon bar, which is directly below the Menu bar.

NOTE: You can execute only a certain part of the written code. To do so, highlight the code that you would like to execute, and then click the **Run** icon.

Troubleshooting Tips

As a general rule, you should make a habit of checking the Log panel after you executed the code. In most cases, a successfully executed code will result in blue messages in the Log panel, as well as a message that indicates that the analysis was successfully completed.

- The Log panel has the following color scheme:

BLUE	→	Successfully executed analyses
RED/BURGUNDY	→	Errors and/or Warnings
GREEN	→	Warnings

- Check if you have placed a semicolon at the end of *every* line of code.
- Check for incorrectly spelled code (e.g. options, dataset name, variable names).
- Check that you have included a snippet of code with the `run;` statement.

Calculating Summary Statistics

In almost all cases, one of the first (and very useful) steps that you want to take when dealing with data analysis is to generate summary statistics. These often include the mean (average), minimum, maximum, number of observations, number of missing observations, standard deviation, and others. With just the summary statistics, you are able to gain a general overview of the data's characteristics.


Summary Statistics Using SAS

To calculate summary statistics using SAS, you will use the MEANS procedure. In the Editor panel, the MEANS procedure can be coded as follows:

```
proc means data=dataset_name options ;  
run;
```

The *options* designate which summary statistics to produce. To produce the mean (average), minimum, and maximum for the wheat prices, you can use the following code:

```
proc means data=wheat_data mean min max ;  
var wheat_p;  
run;
```

After you have written the above code in the Editor panel, click the **Run** icon . The Output panel will automatically be displayed with the resulting summary statistics.

In addition to the mean, minimum, and maximum, you can produce other summary statistics by specifying additional options. The following options can be specified:

Options for PROC MEANS	
Command	Resulting Summary Statistic
mean	Mean (average)
min	Minimum
max	Maximum
n	Number of observations
std	Standard deviation

Implementing Graphical Analysis

Although analytical summary statistics are extremely useful for providing an overview of the data, such an overview may not reveal some additional important information, which may be relevant. A visual representation of the dataset can provide another perspective on examining the data. By plotting and examining the graphical representation, you can observe important factors such as trends, outliers, and patterns.


Graphical Analysis Using SAS

To produce a plot of the data series, you will use the SGPLOT procedure. In the Editor panel, the SGPLOT procedure can be coded as follows:

```
ods graphics on ;
proc sgplot data=dataset_name ;
series y=price_variable x=time_variable ;
run;
ods graphics off ;
```

To generate the plot of the wheat prices over time, you can use the following code:

```
ods graphics on ;
proc sgplot data=wheat_data ;
series y=wheat_p x=date ;
run;
ods graphics off ;
```

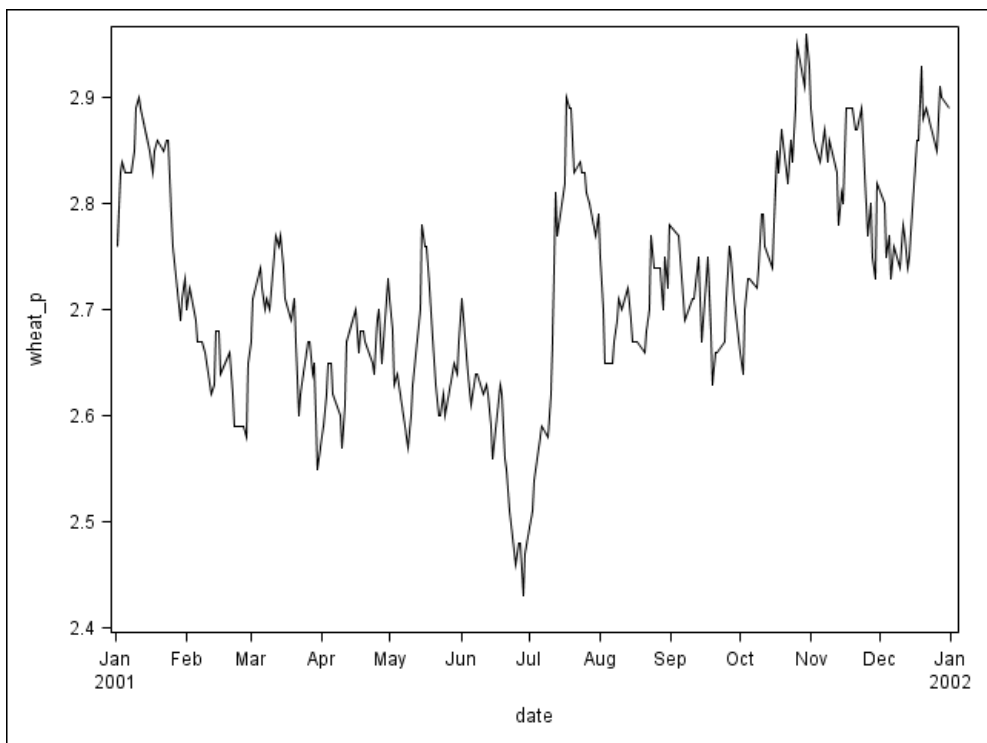
After you have written the above code in the Editor panel, click the **Run** icon .

Retrieving the Resulting Plot

To retrieve the resulting plot, you should do the following:

1. In the Explorer panel, click the **Results** tab at the bottom of the Explorer panel.
2. Double-click the bottom-most icon “Sgplot: The SAS System.” The “Sgplot: The SAS System” folder expands.
3. Double-click the icon “The Sgplot Procedure.” “The Sgplot Procedure” folder expands.
4. Double-click the picture icon “The Sgplot Procedure.” A picture viewer opens with the plot.

For the wheat data example, the plot of the wheat price is as follows:



Code Implemented in Class

```
/* I. Specifying a library name */
libname myLib "C:\Users\anton.bekkerman\Documents\";

/* II. Importing data */

/* Method 1: Use the SAS wizard to import Excel, CSV, Access,
  DBA, etc. data files */

/* Method 2: Use code to import Excel, CSV, Access, etc. data files */
/* Example 1: Importing Excel files */
PROC IMPORT OUT= MYLIB.DT1
            DATAFILE= "C:\Users\anton.bekkerman\Documents\Classes\
            ECNS561\SAS Lab\dt1.xls"
            DBMS=XLS REPLACE;
            GETNAMES=YES;
GUESSINGROWS=50;
RUN;
/* Example 2: Importing CSV files */
PROC IMPORT OUT= myLib.dt1
            DATAFILE= "C:\Users\anton.bekkerman\Documents\Classes\
            ECNS561\SAS Lab\dt1.csv"
            DBMS=CSV REPLACE;
            GETNAMES=YES;
            DATAROW=2;
            GUESSINGROWS=50;
RUN;

/* III. Creating a data set by entering values */

data myLib.dt1;
input record year farm $ acres chickens $2.;
label record="Farm Number"
       year="Year"
       farm="Farm Name"
       acres="Planted Acres"
       chickens="Number of chickens"
;
datalines;
2 1999 HappyFam 554 44
1 1999 MmDonuts 334 32
3 1999 NoRecord . .
5 1999 TinyCow 1234 56
```

```

8 1999 Wheater 442 12
14 1999 Deere 994 72
1 2000 MmDonuts 324 41
2 2000 HappyFam 604 35
3 2000 NoRecord . .
5 2000 TinyCow 1532 76
8 2000 Wheater 204 9
14 2000 Deere 763 63
;
run;

```

```

/* IV. Basic data manipulation */

```

```

/* a. Sorting data */

```

```

/* 1. Sort data by a variable in ascending order (lowest is first) */

```

```

/* Example: Sort farms such that those with lowest acreage are first */
proc sort data=myLib.dt1;
by acres;
run;

```

```

/* 2. Sort data by a variable in descending order (highest is first) */

```

```

/* Example: Sort farms such that those with highest acreage are first */
proc sort data=myLib.dt1;
by descending acres;
run;

```

```

/* 3. Sort data by more than one variables. That is, sort within subgroups */

```

```

/* Example: Sort farms such that those with lowest acreage in each year
are first */
proc sort data=myLib.dt1;
by year acres;
run;

```

```

/* b. Removing/keeping certain observations */

```

```

/* 1. Remove specific observations */

```

```

/* Example: Remove farms that are named "Wheater" */
data myLib.dt2;

```

```

set myLib.dt1;
if farm="Wheater" then delete;
run;

/* 2. Remove observations according to a criteria */

/* Example: Remove all farms that have acreage above or equal to 1,000 */
data myLib.dt2;
set myLib.dt1;
if acres >= 1000 then delete;
run;

/* 3. Keep observations according to a criteria */

/* Example: Keep only farms that have acreage between 300 and 900 acres */
data myLib.dt2;
set myLib.dt1;
if 300 < acres <= 900;
run;

/* 4. Remove missing observations */

/* Example: Remove farms that have a missing observation for acres */
data myLib.dt2;
set myLib.dt1;
if acres = . then delete;
run;

/* c. Manipulating variables (columns) */

/* 1. Keep only certain variables from an original data set */

/* Example: Keep only the year and farm name variables */
data myLib.dt2;
set myLib.dt1(keep=year farm);
run;

/* 2. Delete certain variables from the original data set */

/* Example: Drop the farm number and acreage variables */
data myLib.dt2;
set myLib.dt1(drop=record acres);
run;

```

```

/* 3. Rename variable names */

/* Example: Rename the farm number variable from "record" to "number" */
data myLib.dt2;
set myLib.dt1(rename=(record=number));
run;

/* 4. Convert a character variable into a numeric */

/* Example: Convert the character variable describing the number of
chickens on a farm into a numeric format */
data myLib.dt2;
set myLib.dt1;
chickN = input(chickens, 8.);
drop chickens;
label chickN="Number of chickens";
rename chickN=chickens;
run;

/* V. Advanced data manipulation */

/* First, let's create four data sets for us to use */

data myLib.dta1;
input record year farm $ acres chickens $2.;
label record="Farm Number"
      year="Year"
      farm="Farm Name"
      acres="Planted Acres"
      chickens="Number of chickens"
;
datalines;
2 1999 HappyFam 554 44
1 1999 MmDonuts 334 32
3 1999 NoRecord . .
5 1999 TinyCow 1234 56
8 1999 Wheater 442 12
14 1999 Deere 994 72
;
run;

data myLib.dta2;

```

```

input record year farm $ acres chickens $2.;
label record="Farm Number"
  year="Year"
  farm="Farm Name"
  acres="Planted Acres"
  chickens="Number of chickens"
;
datalines;
2 2000 HappyFam 554 35
1 2000 MmDonuts 324 41
3 2000 NoRecord . .
5 2000 TinyCow 1532 76
8 2000 Wheater 442 9
14 2000 Deere 763 63
;
run;

```

```

data myLib.dta3;
input record year farm $;
label record="Farm Number"
  year="Year"
  farm="Farm Name"
;
datalines;
2 1999 HappyFam
1 1999 MmDonuts
3 1999 NoRecord
5 1999 TinyCow
8 1999 Wheater
14 1999 Deere
1 2000 MmDonuts
2 2000 HappyFam
3 2000 NoRecord
5 2000 TinyCow
8 2000 Wheater
14 2000 Deere
;
run;

```

```

data myLib.dta4;
input farm $ acres chickens $2.;
label farm="Farm Name"
  acres="Planted Acres"

```

```

    chickens="Number of chickens"
;
datalines;
HappyFam 554 44
MmDonuts 334 32
NoRecord . .
TinyCow 1234 56
Wheater 442 12
Deere 994 72
MmDonuts 324 41
HappyFam 604 35
NoRecord . .
TinyCow 1532 76
Wheater 204 9
Deere 763 63
;
run;

/* a. Setting one data set into another */

/* Example: Set the dataset "dta2" onto "dta1" */
data myLib.dta5;
set myLib.dta1 myLib.dta2;
run;

/* b. Merging data sets together */

/* Example: Merge data sets "dta3" and "dta4" together by farm name */
proc sort data=myLib.dta3;
by farm;
run;
proc sort dadta=myLib.dta4;
by farm;
run;

data myLib.dta5;
merge myLib.dta3 myLib.dta4;
by farm;
run;

/* c. Append data sets */

/* Example: Append data set "dta2" to "dta1" */

```

```

proc append base=myLib.dta1
data=myLib.dta2
force;
run;

/* d. Exporting a SAS data set */

/* Method 1: Use the SAS wizard to export a data set */

/* Method 2: Code the export of a data set */
PROC EXPORT DATA= WORK.WEIGHT_IML
      OUTFILE= "C:\Users\anton.bekkerman\Documents\Classes\
      ECNS561\SAS Lab\dt1_exp.csv"
      DBMS=CSV REPLACE;
      PUTNAMES=YES;
RUN;
/* VI. Exploring and summarizing the data */

/* a. Producing basic summary statistics about the data set */

/* Example 1: Default summary statistics about all variables in a data set */
proc means data=myLib.dt1;
run;

/* Example 2: Default summary statistics about only specific variables */
proc means data=myLib.dt1;
var acres;
run;

/* Example 3: Selected summary statistics */
proc means data=myLib.dt1 n nmiss mean median std;
run;

/* b. Frequency statistics */

/* 1. Basic frequency and additional statistics */
proc univariate data=myLib.dt1;
run;

/* 2. Descriptive frequency statistics */
proc freq data=myLib.dt1;
tables farm acres;
run;

```

```

/* c. Visualizing the data -- graphics */

/* 1. Producing a basic scatter plot of the data */

/* Example: Seeing acreage as a function of farm number */
ods graphics on;
proc sgplot data=myLib.dt1;
scatter x=record y=acres;
run;
ods graphics off;

/* 2. Producing a series plot (i.e. connecting the dots) */

/* Example: Seeing the number of holidays in Spain taken by US residents */
ods graphics on;
proc sgplot data=sashelp.tourism;
series x=year y=vsp;
run;
ods graphics off;

/* 3. Producing a bar graph */
/* Example: Bar chart of acres on farms in 1999 */
ods graphics on;
proc sgplot data=myLib.dt1;
where year=1999;
vbar farm / response=acres;
run;
ods graphics off;

/* Example: Bar chart of comparing acres in 1999 and 2000*/
data myLib.dt1_graph;
set myLib.dt1;
if year = 1999 then acres99 = acres;
if year = 2000 then acres00 = acres;
label acres99 = "Acres in 1999"
      acres00 = "Acres in 2000";
run;
ods graphics on;
proc sgplot data=myLib.dt1_graph;
yaxis label = "Acreage";
vbar farm / response=acres99;
vbar farm / response=acres00

```

```

barwidth=0.5
transparency=0.2;
run;
ods graphics off;

/* 4. Producing a histogram and density curve */

/* Example: Distribution of acreage */
ods graphics on;
proc sgplot data=myLib.dt1;
  histogram acres;
  density acres / type=normal;
  density acres / type=kernel;
run;
ods graphics off;

/* VII. Introduction to SAS matrix language */

/* Entering into IML (interactive matrix language) mode */
proc iml;

/* Set option to display row and column numbers */
reset autoname;

/*Construct a vector; columns separated by space, rows separated by "," */
v1 = {1 2 3 4};
print v1;

/*Construct a matrix; columns separated by space, rows separated by "," */
m1 = {1 2, 3 4};
print m1;

/* Change a value in a vector or a matrix */
/* Example: Change the value of the second column in the vector v1 */
v1[,2] = 5;
print v1;

/* Example: Change the value of the second row, first column in matrix m1 */
m1[2,1] = 5;
print m1;

/* Construct an identity matrix; I(num columns) */

```

```

i1 = i(2);
print i1;

/* Construct a matrix of all ones; J(rows, cols, value) */
ones1 = j(2,2,1);
print ones1;

/* Transpose a vector or matrix; t(vector name) or (vector name)' */
v1t = t(v1);
m1t = m1';
print v1t, m1t;

/* Inverse of square matrix; inv(matrix name) */
m1inv = inv(m1);
print m1inv;

/* Adding or subtracting a scalar */
v2 = v1 - 1;
m2 = m1 - 1;
print v2, m2;

/* Multiplying/dividing by scalars */
v3 = v1 # 2;
m3 = m1 # 2;
print v3, m3;

/* Raising to a power */
v4 = v1 ## 2;
m4 = m1 ## (0.5);
print v4, m4;

/* Vector / matrix addition/substraction (must be the same dimensions) */
v5 = v1 + v1;
m5 = m1 + m1;
print v5, m5;

/* Vector/matrix multiplication (must be appopriate dimensions) */
v6 = v1 * v1';
m6 = m1 * m1';
print v6, m6;

/* Determine the rank of a matrix */
m1rank=round(trace(ginv(m1)*m1));

```

```

print m1rank;

/* Read a SAS data set into a matrix */
/* 1. Specify which data set you wish to import into IML */
use sashelp.bweight;

/* 2. Specify that you want to read all of the variables into the matrix */
read all into bweight;
print bweight;

/* 3. Read only specific variables and only a range of observations
into a matrix */

/* Example: Read only the first 100 observations of the variables
"weight" "black" and "married" */
read point (1:100) var {weight black married} into bweight;
print bweight;

/* Summary statistics */
summary var {weight black married} stat{mean std min max} opt{save};
print weight, black, married;

/* Export matrices into SAS data sets */
cols = {"weight" "black" "married"};
create weight_ims from bweight[colname=cols] ;
append from bweight;

/* Exit from IML */
quit;

```