

# AN INTRODUCTION TO USING SAS <sup>1</sup>

SAS is a packaged statistical program, like SPSS, that can be extremely useful to social scientists. We hope this introduction will start you on the road to data analysis with SAS. This document is designed as an introduction to SAS, and will not substitute for the SAS manuals. We assume that you are somewhat familiar with using the UNIX operating system and editing files. Also see the document "Using SAS on the UNIX" for information about using SAS on the UNIX.

## I. CREATING AND RUNNING A SAS PROGRAM

To create a SAS program, use your favorite editor on the UNIX to write a SAS program, for example "dem1040.sas". While it is not required in the UNIX operating system, the convention is to name SAS programs with the second name of ".sas". To run a SAS program, from the command line, you type the command:

```
sas dem1940 & (Substitute the filename of your program for dem1940. The & makes it so that the job runs in background, allowing letting you to issue other commands while the job runs.)
```

Remember that the UNIX is case sensitive. The "sas" command must be in lower case, and the program name must match the case of the program you are running. Within the SAS program, only UNIX commands and file description lines are case sensitive, SAS commands are not.

When you run a SAS program, two output files will usually be written to the directory from which you run the program. Each will have the first name of the file containing the program you ran. One will have the second name of ".log", and will list your commands, and SAS's response to them. The second file will have the second name of ".lst", will have any output you asked for. Be sure to look over both the log and the lst file.

## II. CREATING A SAS DATA SET, READING AND DEFINING DATA

If you are going to create a data set, you begin with a DATA statement. You can provide a name for the data set, or you can allow SAS to assign a name to it. It is a good idea to assign it a name yourself, to make it easier to access it later in the program if you want to. In the following example of a data statement, I am assigning the name STATE.

```
DATA STATE;
```

The first step toward data analysis is to read data. Data can come in different forms. In many cases, you will start out by reading a raw data file. For example:

1	35	33	44	ARIZONA	45
3	44	33	55	MAINE	-9
4	32	33	44	OREGON	23
5	34	44	33	GEORGIA	34
2	32	77	33	WASHINGTON	-9

In the above example, each line, or row, represents a case. There are six variables for each case. SAS reads raw data with an INPUT statement. The input statement gives SAS information about the variables it is going to read. Variable names can be up to eight characters long. The above data could be read as free-field or fixed-field. A data can be read as free-field if there is at least one space (sometimes two) between each variable, and missing data are not represented with blanks.

The above data could be read with the following free-field input statement:

```
INPUT ID V1-V3 NAME $20. V4;
```

When SAS encounters a space, it realizes that it has come to the end of a variable, and goes on to read the next in the variable list. Unless SAS is told otherwise, SAS expects numbers. NAME is a character variable. You tell SAS this with a dollar sign. When reading character variables, unless it is told otherwise, SAS makes the length of the variable the same as that

---

<sup>1</sup>Prepared by Patty Glynn and Stewart Tolnay and distributed courtesy of the Center for Social and Demographic Analysis, University at Albany, State University of New York. March 21, 1994. Modified for use on UNIX rather than IBM, December 21, 1999.

of the first case it encounters. If the first case it encounters has a length of 10, that is the length allowed for the variable. Cases with more letters will be truncated. The \$20. tells SAS to allow up to 20 spaces for the character variable.

What if NEW YORK were one of the states in this list? With the input statement above, SAS would read NEW as the state name, and try to read YORK as V4. But, you can tell SAS that there might be **one** space in a character variable by using a "&". You could use the following free-field input statement to read the data correctly.

```
INPUT ID V1-V3 NAME & $20. V4;
```

This tells SAS that if **one** space occurs in the first 20 characters after it starts reading NAME, that it does not mean it is the end of the variable. If two consecutive spaces are encountered, SAS will go on to read the next variable.

The above data could also be read as fixed-field with the following input statement.

```
INPUT ID 1-2 V1 9-10 V2 18-19 V3 27-28 NAME $ 33-45 V4 46-47;
```

This tells SAS that it will find the information about ID in columns 1 and 2, about V1 in columns 9-10, and so on. With fixed-field input, missing data can be represented as blanks, and more than one space can be embedded in character values.

Most data sets are fixed-field without spaces between values, requiring less space to store the data.

Before current technology, data and programs were stored on cards with holes punched in them. Cards could only hold 80 columns of data. As a hold-over from that time, each row of data is sometimes referred to as a card. Another hold-over from that time is that data are often stored with a width of just 80 columns. **And, SAS programs cannot have instructions beyond the 80th column.**

Large data sets often have more than one record of data for each case or observation. SAS needs more information when reading these data sets. The input statement must specify the card number for each variable being read, and must tell SAS how many cards there are for each case. For example, the ICPSR dataset, Demographic Characteristics of the Population of the U.S., 1930-1950, which has information about population and migration for counties and states, has 55 records for each case. Following is part of the input statement that reads that data set.

```
INPUT
#1 M001 3-4 M002 5-8 M003 9-12 M004 $ 13-32 M005 $ 33-64
  M006 $ 65 M007 $ 66 M008 67 M009 68 M010 69-75
#2 M011 13-19 M012 20-26 M013 27-33 M014 34-40 M015 41-47
  M016 48-54 M017 55-61 M018 62-68 M019 69-75
#3 M020 13-19 M021 20-26 M022 27-33 M023 34-40 M024 41-47
  M025 48-54 M026 55-61 M027 62-68 M028 69-75
#4 M029 13-19 M030 20-26 M031 27-33 M032 34-40 M033 41-47
  M034 48-54 M035 55-61 M036 62-68 M037 69-75
#54 M479 13-19 M480 20-26 M481 27-33 M482 34-39 .2
  M483 40-45 .2 M484 46-51 .2 M485 52-57 .2
  M486 58-63 .2 M487 64-69 .2 M488 70-75 .2
#55 ;
```

In the above example, there are 55 records for each case, but variables are being read from only the first 4 cards, and card 54. The #55 tells SAS that there are 55 cards for each case. You must tell SAS how many cards there are per case, or the data will not be read properly.

Notice the ".2" in the part of the input statement that defines card 54. If there are not embedded decimals in the data, and with many data sets there are not, you must tell SAS how many digits should be to the right of the decimal. "M486 58-63 .2" tells SAS to read the variable M486 in columns 58 to 63 (of card 54) and that 2 digits should be to the right of the decimal. So, the digits "123456" would be read as "1234.56".

Now you know something about how to tell SAS to read data. You must also tell SAS where to find the data. Data can be stored in a separate file on disk, or, if the data have 80 or fewer columns of data, within the program file. It is usually not efficient to store data within a program. But, when you want to do it this way, after your input statement and other data

definition statements which we will discuss later, you have a CARDS statement, followed by the data. After the last line of data, you must put a semicolon. For example:

```
DATA STATE ; INPUT ID V1-V3 NAME & $20. V4;
CARDS;
1      35      33      44      ARIZONA      45
3      44      33      55      MAINE         -9
4      32      33      44      OREGON        23
5      34      44      33      GEORGIA       34
2      32      77      33      WASHINGTON    -9
;
```

An "INFILE" statement tells SAS that you want it to read raw data from disk, and tells it where it will find it. The infile statement can look like: **INFILE '/full/path/filename';** Or, the INFILE statement can refer to a FILENAME statement, as below.

```
FILENAME INDD1 '/full/path/state.data' ;
INFILE INDD1;
```

In the above example, INDD1 is defined with the FILENAME statement. When SAS encounters the arbitrary word (which must be 8 or fewer characters) INDD1, it knows that we are referring to state.data. By using a FILENAME statement, you can give SAS more information about the data (which you must do when you are reading data from tape). (For more information about using tapes, see "USING TAPES ON THE IBM MAINFRAME AT SUNY-A" which is also distributed by CSDA.) Also, by using a filename statement, it is possible to read a data set that is stored in compressed format.

```
filename inz1 PIPE 'zcat /full/path/raw.data.gz' lrecl = 80 ;
data state; INFILE inz1 ;
```

You may have noticed that SAS statements end with semicolons. Semicolons are the major punctuation used in SAS. They indicate when statements end.

SAS uses asterisks for comments. A comment is a non-executable statement that you can use to document your programs. Comments must end with a semicolon. Comments make your programs more readable, and will help you remember what your programs are doing, and why.

An OPTIONS statement allows you to give special instructions to SAS. For example, you can ask it to write no more than 72 characters per line on output (the .lst file), so that it will be easier to read on the screen. You can also ask it to compress files, so that you will require less disk space in running your programs and storing SAS system files. Following is an example of an options card.

```
OPTIONS LINESIZE=72 COMPRESS=YES;
```

So, let's put this all together and create a SAS program that reads the fictional sample data above.

```
* SAMPLE SAS. THIS IS A SAMPLE SAS PROGRAM THAT SHOWS HOW TO READ;
* NUMERIC AND CHARACTER VARIABLES IN A FREE-FIELD FORMAT. ;
OPTIONS LINESIZE=72 COMPRESS=YES;
filename inz1 PIPE 'zcat /full/path/state.data.gz' lrecl = 80 ;
DATA STATE; INFILE inz1;
INPUT ID V1-V3 NAME & $20. V4;
```

As mentioned above, variable names cannot be more than 8 characters long. It is difficult to get a meaningful variable name in 8 letters. However, SAS allows you to assign labels of up to 40 characters. You do this with a LABEL statement. For example, the following LABEL statement goes with the demographic data mentioned above. Notice that the labels are enclosed in single quotes. Regular quotes can also be used.

```
LABEL M001='M001=ICPR STATE CODE' M002='M002=UNIQUE ID'
M003='M003=SEQUENTIAL ID' M004='M004=STATE NAME'
M007='M007=940-950 CNTY BOUNDARIES' ;
```

### III. TRANSFORMING DATA

Now that you know how to read raw data, you need to know what you can do with it. We will use the demographic data as an example.

In many data sets there are codes for missing data. For example, in the demographic data set described above, 9999999 is the missing value for many variables. It is important that you tell SAS that these values mean missing data. For numeric data, a period means missing data to SAS. The following statements will change the data so that SAS will know they are missing. Missing data are not used in computing means, sums, regressions, etc. Failing to define values as missing can result in wrong answers to the questions you ask.

```
IF M001 = 99 THEN M001 =.; IF M002 = 9999 THEN M002 =.;  
IF M003 = 9999 THEN M003 =.; IF M008 = 0 THEN M008 =.;  
IF M009 = 0 THEN M009 =.; IF M010 =9999999 THEN M010 =.;
```

ARRAY statements, with a DO OVER statement can also be used to change values to missing. If a long list of variables has the same missing value, an ARRAY statement can be very useful. For example:

```
ARRAY MISS1 M011 M012 M013 M014 M015 M016 M017 M018 M019 M020  
M021 M022 M023 M024 M025 M026 M027 M028 M029 ;  
DO OVER MISS1; IF MISS1=9999999 then MISS1=.; END;
```

The above statements changed the values for 23 variables from 9999999 to missing. Instead of 23 lines of code, only 3 were required. MISS1 is the arbitrary name of the array (must be 8 or fewer characters, and not a variable or array name used elsewhere). Within the DO OVER loop, each variable in the array list is processed. There must be an END statement at the end of the DO OVER loop.

Suppose you want to know the percent of males that are 20 to 29 in each county. You could create that variable with the following statements.

```
MALE2029=( (M015+M016)/M010)*100;  
LABEL MALE2029='% OF MALES THAT ARE AGED 20-29';
```

SAS also has a SUM function. The statement MALE2029=(SUM(M015,M016)/M010)\*100; could also be used. But, if there is missing data for M015 or M016, the results you get using a plus sign and the SUM function may not be the same. With the SUM function, SAS will give you a non-missing answer if any of the values in the parentheses have non-missing values. However, using the plus sign, SAS will give you a non-missing answer only if all of the variables added have non-missing values. This can be a very important difference. In this case, we would want to use the plus sign. Otherwise, we would have inaccurately low values for counties that were missing information for either M015 or M016.

SAS has many useful functions available for both numbers and characters. Look in a SAS manual for information about others.

Arrays can be very useful in creating new variables, as well as transforming old ones. The following statements will calculate what percent of the population that each age group comprises, and assign the values to new variables called P011 to P029. The DO OVER loop processes both arrays. When it is working with M011, it will work with P011. When it is working with M012, it will work with P012, etc. If the variables are not listed in the same order, the results would not be correct.

```
TOTPOP=M010;  
ARRAY ORIG M011 - M029 ;  
ARRAY PERPOP P011 - P029 ;  
DO OVER ORIG;  
PERPOP=(ORIG/TOTPOP)*100;  
END;
```

### IV. KEEP AND DROP STATEMENTS, AND SUB-SETTING IF STATEMENTS

Often you will want to work with only some of the variables, and some of the cases. SAS programs will take longer than necessary to run, and will require more disk resources than necessary if there are more cases or variables than necessary in your

data set. A KEEP or DROP statement can be used to prune the number of variables. For example, suppose you were using the Demographic Characteristics data, and were only interested in some of the variables that pertained to 1940. You could use the following statement to keep the variables that identify each county, and that pertain to 1940.

```
KEEP M001 M002 M003 M004 M005 M006 M007 M008 M009 M096 - M112 ;
```

To reduce the number of cases, you can use a sub-setting IF statement. For example, suppose you were only interested in the counties in Georgia. You could select these counties with the statement:

```
IF M001=44 ;
```

Or, if you wanted only counties in the Georgia, Florida, Alabama, and Tennessee, you would have the statement:

```
IF M001=44 OR M001=43 OR M001=41 OR M001=54;
```

Please see the example that begins on page 8 to see how and where keep, drop, and sub-setting if statements can be used.

## **V. SAS PROCEDURES or "PROC"s**

Once you have read your data, and created new variables, you are ready to do something with them. After you use a SAS procedure, the SAS dataset can no longer be altered. You cannot change the data set by creating new variables, or with "keep", "drop", or subsetting "if" statements after you have used a procedure, without a new DATA statement. Forgetting to put in a semicolon is probably the most common mistake in SAS. Trying to modify the data after using a PROC is probably the second most common mistake. If you want to make changes in a data set after running a procedure, you must precede the modification with a DATA step (discussed above).

SAS has many "PROCEDURES". These are sets of instructions programmed within SAS. For example, there are procedures for sorting data, printing data, calculating means, T-tests, anovas, correlations, regressions, logistic regressions, frequencies, crosstabs, and estimating structural equation models. Before packaged programs such as SAS and SPSS, it could take weeks or months to calculate a multiple regression equation. Now, it can take a few minutes to write the program, and seconds to get the results.

When invoking a SAS procedure, you always start with "PROC". For example, if you wanted to know about the means, medians, quartiles, skew, number of cases, minimum and maximum values and other information about some variables, you would use PROC UNIVARIATE. PROC MEANS is another procedure that gives less information about individual variables in a more concise format.

Refer to the last pages of this handout for an example of a program and results of some SAS procedures.

## **VI. CREATING PERMANENT SAS DATA SETS**

All of the SAS data sets in the above examples would disappear after the job has finished running. They are temporary. But, it is often useful to create SAS data sets (or systems file) that you can access later. Reading raw data is time consuming for SAS with large data sets. By creating a permanent SAS file, you can save yourself time and computer resources. When you access the permanent SAS systems file, it will know everything that you told SAS in the DATA step that created the file. If you gave the variables labels, it will know the labels. If you transformed variables, or created new variables, those variables will be stored. If you told SAS what values are missing, it will remember what you told it.

If you want to create a permanent SAS systems file, in the DATA statement, you provide a TWO LEVEL name. For example MYLIB.DEM1940. Note the LIBNAME statement. The LIBNAME refers to a directory where SAS systems files will be stored. In this case, a file '/full/path/directory/dem1940.ssd01' would be created. On UNIX, SAS data systems files have the second name of .ssd01. You do not need separate LIBNAME statements for each SAS systems file. The specific member of the library is defined with the second level name.

```
* DEM1940 SAS. THIS PROGRAM CREATES A SAS FILE CALLED DEM1940;  
* WHICH HAS SELECTED DEMOGRAPHIC INFORMATION ON GEORGIA;  
* COUNTIES FOR 1940. ;  
filename inz1 PIPE 'zcat /full/path/dem1940.data.gz' lrecl = 80 ;
```

```

LIBNAME mylib '/full/path/directory'; * do not include filename ;
OPTIONS LINESIZE=72 COMPRESS=YES;
DATA mylib.dem1940; INFILE inz1;
INPUT #1 M001 3-4 M002 5-8 M003 9-12 M004 $ 13-32 M005 $ 33-64
      M006 $ 65 M007 $ 66 M008 67 M009 68
      #11 M096 41-47 M097 48-54 M098 55-61 M099 62-68 M100 69-75
      #12 M101 13-19 M102 20-26 M103 27-33 M104 34-40 M105 41-47
      M106 48-54 M107 55-61 M108 62-68 M109 69-75
      #13 M110 13-19 M111 20-26 M112 #55 ;

LABEL
M001='M001=ICPR STATE CODE' M002='M002=UNIQUE ID'
M003='M003=SEQUENTIAL ID' M004='M004=STATE NAME'
M005='M005=COUNTY NAME' M006='M006=930-940 CNTY BOUNDARIES'
M007='M007=940-950 CNTY BOUNDARIES'
M008='M008=SPECIAL COUNTY INFORMATN'
M009='M009=NEGATIVE ESTIMATE INDCTR'
M096='M096=940 TTL MALES, ALL AGES'
M097='M097=940 TTL MALES, <5' M098='M098=940 TTL MALES, 5-9'
M099='M099=940 TTL MALES, 10-14' M100='M100=940 TTL MALES, 15-19'
M101='M101=940 TTL MALES, 20-24' M102='M102=940 TTL MALES, 25-29'
M103='M103=940 TTL MALES, 30-34' M104='M104=940 TTL MALES, 35-39'
M105='M105=940 TTL MALES, 40-44' M106='M106=940 TTL MALES, 45-49'
M107='M107=940 TTL MALES, 50-54' M108='M108=940 TTL MALES, 55-59'
M109='M109=940 TTL MALES, 60-64' M110='M110=940 TTL MALES, 65-69'
M111='M111=940 TTL MALES, 70-74' M112='M112=940 TTL MALES, 75+' ;
* CREATE VARIABLES FOR MERGING LATER;
STATEID=M001; COID=M002;
* KEEP ONLY GEORGIA;
IF STATEID=44 ;
IF M001 = 99 THEN M001 =.; IF M002 = 9999 THEN M002 =.;
IF M003 = 9999 THEN M003 =.; IF M008 = 0 THEN M008 =.;
IF M009 = 0 THEN M009 =.;
ARRAY MISS1 M096 - M112 ;
DO OVER MISS1;
IF MISS1=9999999 then MISS1=.;
END;
PROC SORT; BY STATEID COID ;
PROC MEANS;

```

## VII. ACCESSING PERMANENT SAS DATA SETS

A **SET** statement (or merge statement, as discussed next) is used to access a SAS file. **SET** and **MERGE** can be used with either permanently stored or temporary files. The following program would access the permanent SAS file created in the above example.

```

* DEM19402 SAS. THIS PROGRAM READS A SAS FILE CALLED DEM1940;
* WHICH WAS CREATED BY THE PROGRAM DEM1940 SAS.;
* IT HAS INFO ON COUNTIES FOR 1940. ;
LIBNAME mylib '/full/path/directory'; * do not include filename ;
OPTIONS LINESIZE=72 COMPRESS=YES;
DATA DEM40; SET mylib.dem1940;

```

## VIII. MERGING SAS DATA SETS: ADDING MORE VARIABLES

In data analysis, it is common to pull together data from a variety of sources. It is possible to do this with SAS by **MERGING** data sets. It is important to merge correctly. It is preferable if cases have unique identifying numbers. This makes merging easier. Following is a simple example of merging data. Test scores from two exams will be merged.

```

* test12.sas. TEST SCORES FOR EXAMS 1 AND 2;
OPTIONS LINESIZE=72 COMPRESS=YES;
DATA TEST1; INPUT ID SCORE1 NAME & $20.;
CARDS;
1 30 JANE DOE
2 14 JOHN JONES
4 23 SALLY SMITH
5 32 SUSAN WARD
3 35 JIM JOHNSON
;
PROC SORT; BY ID;
DATA TEST2; INPUT ID SCORE2;
CARDS;
1 40
4 45
3 43
5 41
;
PROC SORT; BY ID;
DATA TEST12; MERGE TEST1 TEST2; BY ID ;

```

The dataset TEST12 will have 5 cases, and 4 variables (ID, SCORE1, NAMES, and SCORE2). Note that TEST2 only has 4 cases. John Jones did not take the second test. But, because we merge **BY ID**, the merge will work properly. John Jones will have missing data for SCORE2. If each case did not have a unique identifying number, it would not be possible to merge these two dataset together properly. Note that PROC SORT is used on each data set. The files must be sorted in order of the BY variable in order for the merge to work.

In the Demographic Characteristics dataset, M001 and M002 together, uniquely identify each county. M001 identifies the state, and M002 identifies each county within each state. ICPSR uses these identifying numbers for many of its data sets that have county-level information. The "BY" variables must have the same names in all of the datasets to be merged. When we use county data sets, we always create variables that can be used for merging. Notice that we did so in the job called DEM1940 SAS, with the assignment statements **STATEID=M001; COID=M002;**

Suppose you wanted to merge the file created by the program called DEM1940 with a SAS file that you already created called "relig40.ssd01". "relig40.ssd01" contains information about church membership in 1940. It is also distributed by ICPSR. As with "dem1940.ssd01", you created variables called STATEID and COID, and have already sorted the files by those variables. It is important to **not** have variables with the same name in the files you are merging, **except the variables you are merging by**. Two different variables cannot have the same name in a file, so one would be lost in the merge. The following program would merge the files.

```

* DEMREL40 SAS. THIS PROGRAM MERGES DEMOGRAPHIC AND RELIGION;
* DATA FOR 1940, AND CREATES A SAS FILE CALLED demrel.ssd01 ;
LIBNAME mylib '/full/path/directory'; * do not include filename ;
OPTIONS LINESIZE=72 COMPRESS=YES ;
DATA mylib.demrel40; MERGE mylib.DEM1940 mylib.RELIG40; BY STATEID COID;
PROC CONTENTS POSITION;
PROC CORR; VAR CATH1940 M019;

```

After merging data, it is important to check your SASLOG, and output to make sure that it was done properly. Make sure that the number of cases is correct. Do a PROC CORR (to see a correlation matrix) including variables from both data sets and check the number of observations.

## IX. CONCATENATING SAS FILES: ADDING MORE CASES

Suppose you had a SAS file called "int500.ssd01" with interviews from 500 people. You just completed creating a SAS file called INT200 for another 200 people, and you want to add those cases to your file. All of the people were asked the same questions, and both SAS files have the same variables. The following program would put these two files together.

```

* INT700 SAS. CREATE ONE FILE OUT OF INT500 AND INT200;
LIBNAME mylib '/full/path/directory'; * do not include filename ;
OPTIONS LINESIZE=72 COMPRESS=YES ;
DATA mylib.int700; SET mylib.int500 mylib.int200 ;
PROC CONTENTS POSITION;

```

## X. AN EXAMPLE OF A SAS PROGRAM WITH A VARIETY OF PROCEDURES

Following is the log and lst files of a SAS program that accesses a SAS systems file, selects a subset of cases, creates new variables, keeps a subset of variables, and runs some procedures.

1 The SAS System 15:11 Thursday, October 22, 1998

NOTE: Copyright (c) 1989-1996 by SAS Institute Inc., Cary, NC, USA.

NOTE: SAS (r) Proprietary Software Release 6.12 TS020

Licensed to SUNY ALBANY, Site 0011801008.

This message is contained in the SAS news file, and is presented upon initialization. Edit the files "news" in the "misc/base" directory to display site-specific news and information in the program log.

The command line option "-nonews" will prevent this display.

NOTE: AUTOEXEC processing beginning; file is /malthus1/local/sas612/autoexec.sas.

NOTE: SAS initialization used:

real time 0.390 seconds

cpu time 0.094 seconds

NOTE: AUTOEXEC processing completed.

```
1      * exampl.sas ;
2      * uses msa6054.sas created by /malthus6/csda/icpsr/6054/msa6054.sas;
3      title1 'exampl.sas, uses MSA level 1990 STF3C data' ;
4
5      options compress = yes linesize = 72 ;
6
7      * P8_1 = "RACE[5] White"          ;
8      * P8_2 = "RACE[5] Black"         ;
9      * P8_3 = "RACE[5] American Indian, Eskimo, or Aleu" ;
10     * P8_4 = "RACE[5] Asian or Pacific Islander" ;
11     * P8_5 = "RACE[5] Other race"    ;
12
13     libname da6054 '/malthus6/csda/icpsr/6054' ;
NOTE: Libref DA6054 was successfully assigned as follows:
Engine:          V612
Physical Name:  /malthus6/csda/icpsr/6054
14
15     * access the data and keep only some variables ;
16     * Using a "keep = " on a set statement is efficient;
17     data msa; set da6054.msa6054 ( keep =
18     ANPSADPI MSACMSA P1_1 P8_1 P8_2 P8_3 P8_4 P8_5 ) ;
19
20     * there are no missing values for these variables ;
21     * for these cases so can use the sum function;
22     * Create a variable of total population ;
23     p8tot = sum(P8_1,P8_2,P8_3,P8_4,P8_5) ;
24
25     perwhite = (P8_1 / p8tot) * 100 ;
26     perblack = (P8_2 / p8tot) * 100 ;
27     peramind = (P8_3 / p8tot) * 100 ;
28     perasian = (P8_4 / p8tot) * 100 ;
29     perother = (P8_5 / p8tot) * 100 ;
30
31     label
32     p8tot = 'Total population in MSA'
```

```

33     perwhite = 'Percent White'
34     perblack = 'Percent Black'
35     peramind = 'Percent American Indian, Eskimo, or Aleu'
36     perasian = 'Percent Asian or Pacific Islander'
37     perother = 'Percent Other race' ;
38
39
40     * select only MSAs with 5% or higher asian population ;
41     if perasian ge 5 ;
42
43     * drop variables no longer needed. ;
44     drop P1_1 P8_1 P8_2 P8_3 P8_4 P8_5 ;
45

```

NOTE: The data set WORK.MSA has 13 observations and 8 variables.  
NOTE: Compressing data set WORK.MSA decreased size by 0.00 percent.  
Compressed is 1 pages; un-compressed would require 1 pages.  
NOTE: DATA statement used:

real time	3.070 seconds
cpu time	0.596 seconds

```

46     proc sort; by ANPSADPI ; * ANPSADPI is area name ;
47

```

NOTE: The dataset WORK.MSA has 13 observations and 8 variables.  
NOTE: Compressing data set WORK.MSA decreased size by 0.00 percent.  
Compressed is 1 pages; un-compressed would require 1 pages.  
NOTE: PROCEDURE SORT used:

real time	0.550 seconds
cpu time	0.037 seconds

```

48     proc print label;
49     var ANPSADPI perasian perwhite perblack peramind perother ;
50     format perwhite perblack peramind perasian perother 8.2 ;
51

```

NOTE: Access by observation number not available.  
Observation numbers will be counted by PROC PRINT.  
NOTE: The PROCEDURE PRINT printed page 1.  
NOTE: PROCEDURE PRINT used:

real time	0.050 seconds
cpu time	0.012 seconds

```

52     proc means;
53     var p8tot perwhite ;
NOTE: The PROCEDURE MEANS printed page 2.
NOTE: PROCEDURE MEANS used:

```

real time	0.010 seconds
cpu time	0.006 seconds

NOTE: The SAS System used:

real time	4.200 seconds
cpu time	0.771 seconds

NOTE: SAS Institute Inc., SAS Campus Drive, Cary, NC USA 27513-2414

OBS	Area Name/PSAD Term/Part Indicatr	Percent Asian or Pacific Islander
1	Fresno, CA MSA	8.58
2	Honolulu, HI MSA	63.07
3	Los Angeles--Anaheim--Riverside, CA CMSA	9.22
4	Merced, CA MSA	8.31
5	Modesto, CA MSA	5.09
6	Sacramento, CA MSA	7.75
7	Salinas--Seaside--Monterey, CA MSA	7.78
8	San Diego, CA MSA	7.95
9	San Francisco--Oakland--San Jose, CA CMSA	14.84
10	Seattle--Tacoma, WA CMSA	6.42
11	Stockton, CA MSA	12.44
12	Washington, DC--MD--VA MSA	5.14
13	Yuba City, CA MSA	9.06

OBS	Percent White	Percent Black	Percent American Indian, Eskimo, or Aleu	Percent Other race
1	63.48	4.92	1.13	21.89
2	31.66	3.07	0.47	1.72
3	64.71	8.44	0.60	17.02
4	67.52	4.92	0.94	18.31
5	80.41	1.63	1.18	11.70
6	79.09	6.84	1.23	5.09
7	63.90	6.38	0.88	21.07
8	75.08	6.30	0.86	9.80
9	69.42	8.56	0.65	6.52
10	86.54	4.76	1.29	1.00
11	73.52	5.59	1.21	7.24
12	65.76	26.56	0.31	2.23
13	77.76	2.75	2.15	8.28

Variable	Label	N	Mean
P8TOT	Total population in MSA	13	2635251.77
PERWHITE	Percent White	13	69.1418965

Variable	Label	Std Dev
P8TOT	Total population in MSA	4000562.85
PERWHITE	Percent White	13.4015939

Variable	Label	Minimum
P8TOT	Total population in MSA	122643.00
PERWHITE	Percent White	31.6619451

Variable	Label	Maximum
P8TOT	Total population in MSA	14531529.00
PERWHITE	Percent White	86.5352514