

More on SAS Programming and Some Applications

Although several approaches are possible for introducing the SAS language, in presenting the material in Chapter 1 in this book, the authors have consciously avoided a cookbook approach. The earlier students encountered the concepts of pointers and program data vectors, for example, the better their understanding of the working of the SAS data step. Without a basic understanding of the flow of operations in the data step, they will be not be prepared to use the data step effectively. From experience, it has been observed that this technique is more effective in getting the students to a higher point in the learning curve much earlier than using a cookbook approach. Having mastered the material in Chapter 1, readers will be ready to examine the use of SAS for data analysis in greater detail. In this chapter, several SAS procedures are used to illustrate the use of some statements common to many SAS procedures as well as a few that are specific to each procedure. To begin Chapter 2, some useful SAS statements available in both data and proc steps not introduced previously are discussed in detail.

2.1 More on the DATA and PROC Steps

In the previous chapter, presentation of many important aspects of both the data and proc steps were deferred in order to keep the material presented in those sections, to some degree, less cluttered. Some of these topics are covered in detail in this section. Many readers who are already familiar with SAS may have observed that in previous examples, the raw data were input as instream data when the content of the data sets called for the data to be read from external text files. This choice was made because it has been the experience of the authors that the introduction of the `infile` statement, the primary tool for accessing data from external files, at an early stage impedes the beginning SAS user from understanding the basic data step operations. This is due to the fact it makes it more difficult to follow the data step processing clearly when the raw data lines are not in direct view of the user. Also, some users

have a tendency to confuse the external data file with the SAS data set being created. Thus, SAS data sets and their creation was introduced early.

2.1.1 Reading data from files

The **infile** statement is primarily used to specify the external file containing the raw data, but it includes options to allow the user more control during the process of transferring data values from the raw data file into a SAS data set. For example, the user may use an option available in the **infile** statement to change the “delimiter”, used by the *list input style* for reading data with the input statement, from a blank space to another character such as a comma. Another option allows the user to be given control when end-of-file is reached when reading external data so that other actions may be initiated before closing the new SAS data set.

The INFILE Statement

In previous SAS examples presented in Chapter 1, the data lines were inserted instream preceded by a **datalines** statement to identify the beginning of the data lines (see SAS Example A1 program in Fig. 1.1 in Chapter 1). The **infile** statement is an executable statement required to access data from an external file. In a SAS data step, it must obviously be present before the input statement because the execution of input statement requires the knowledge of the source of the raw data. The general form of the **infile** statement is

```
INFILE file-spec <options> ;
```

where *file-spec* represents a file specification. In the Windows environment, the file specification is easiest to be given directly as a path name to a file inserted within quotes; for example,

```
infile 'C:\Documents and Settings\...\demogr.data';
```

However, this may become cumbersome if some options are also to be included in the **infile** statement. Thus, it may be convenient to use a *fileref*.

The FILENAME Statement

The nonexecutable **filename** statement associates the physical name and location of an external file with a *fileref*, which is an alias for the file. The *fileref* is then available for use within the current SAS program. Under the Windows environment, a *fileref* is synonymous with the path name of the file. Text files previously saved in a folder can be given a *fileref* by including a **filename** statement in the SAS program. The following statements assign a *fileref* to the file named **demogr.data** and uses it in an **infile** statement:

```
filename mydata 'C:\Documents and Settings\...\demogr.data';
infile mydata;
```

When a `datalines` statement is used to process instream data, SAS automatically assumes an `infile` statement with the file specification `datalines`; thus the `infile` statement is not required, unless the user wants to use an option available on the `infile` statement. In that case, the user must include an `infile` statement even if the data are included instream. An example of the use of an option while reading instream data is

```
filename datalines eof=last;
```

The above option specifies that once the last data line has been processed, the data step is to be continued by transferring control to the SAS statement labeled `last` instead of closing the SAS data set and terminating the data step. For instance, if the last observation has not yet been written to the SAS data set when end-of-file is encountered (for some reason, such as the last data line being incomplete), this allows the user to define how that situation should be handled.

Example 2.1.1

This is a simple conversion of SAS Example A1 displayed in Fig. 1.1 in Chapter 1 to read the raw data set from a file instead from data entered instream. First, suppose that the data set is available as a text file prepared by entering the data lines into a simple text editor such as Notepad (if a word processor is used to enter the data, the user must make sure that the file is saved as a simple text file). Assume the file is named, say, `wages.txt` and is saved in a folder under the Windows environment. The SAS Example A1 program must be modified to access the data from this file as follows:

```
data first ;
infile 'C:\Documents and Settings\...\wages.txt';
input (income tax age state)(@4 2*5.2 2. $2.);
run;
proc print ;
title 'SAS Listing of Tax data';
run;
```

Here the file specification is a quoted string giving the path of the file containing the raw data.

Some Infile Statement Options

There are several `infile` statement options that may be useful for managing the conversion of information in data lines to an observation, such as the `eof=` option discussed earlier or the `n=` option discussed in the Section 1.7.5. They are too numerous to be discussed in detail in this book; however, a few are sufficiently important to be briefly mentioned here. The `delimiter=` or the `dlim=` option allows the user to change the default value of the separator of

data values, when using the the list input style to read data, from a space to the character specified. To read data separated by commas, use

```
infile datalines delimiter=',';
```

If any of the data values contain an embedded comma, this option will not work; instead, the `dsd` must be used:

```
infile datalines dsd;
```

With this option in force, a missing value is assumed if two consecutive commas are detected. When using a list input style if a line contains fewer data values than indicated in the input statement, use the `missover` option to prevent SAS from moving the input pointer to the next line to read the needed values:

```
infile datalines missover;
```

The `missover` option sets the remaining `input` statement variables to missing values. The option `flowover` is the default. The default causes the pointer to move to the next input data line if the current input line is not complete. The options such as `firstobs=` and `obs=` allow the user to access a specified number of data lines beginning from a specified line of data in the external data set. For example, the following processes data lines 20 through 50:

```
infile datalines firstobs=20 obs=50;
```

If `firstobs=` is omitted, SAS will access the first 50 data lines. The `n=` option specifies the number of lines that the pointer can move to in the input buffer using the `#` pointer control in a single execution of the input statement. The default value is 1. See Section 1.7.5 for more details.

2.1.2 Combining SAS data sets

When several data sets are created using multiple sources, they must be combined before a meaningful statistical analysis can be performed. Depending on the structure and the format of the input data sets and those required of the output data set, a variety of methods are available in SAS to form a combined data set. The SAS data step statements `set`, `merge`, and `update` are the primary tools available for combining data sets in a SAS data step. In SAS Example A2 (see Fig. 1.5 in Chapter 1 for the program), the `set` statement was used to illustrate how a SAS data set containing a subset of another SAS data set may be created as follows:

```
data second;
set first;
if age<35 & state='IA';
run;
```

Here an `if` statement was used to select those observations that satisfy the condition specified.

SAS Example B1

In this section an example is used to illustrate the use of the `set` statement to combine two SAS data sets by appending observations from one data set to those of the other. This process is called *concatenation* and allows the combination of several data sets. It is usually practicable when the data sets contain data from similar studies. This implies that the input data sets are expected to contain exactly the same variables (i.e., variables with identical names). It is possible that a few variables are different among some data sets due to decisions taken during the data collection process. If one or more of the data sets contain variables that are not common to all, the combined data set will contain those variables, but with missing values in the observations formed from the data sets that do not contain those variables.

```
data third;
input w 1-2 x 3-5 y 6;
datalines;
211023
312034
413045
;
run;
data fourth;
input x y z;
datalines;
14 5 7862
15 6 6517
16 7 8173
;
run;
data fifth;
set third fourth;
run;
proc print;
title 'Combining SAS data sets end-to-end ';
run;
```

Fig. 2.1. SAS Example B1: Program

Three SAS data sets named `third`, `fourth`, and `fifth` are created in the SAS Example B1 program (see Fig. 2.1), the first two using external data and the other by combining the two SAS data sets previously created. The first data step uses the column input style to create the data set `third` and the second uses the list input style to create the data set `fourth`, both containing **three** observations and **three** variables, respectively (see the abbreviated SAS log in Fig. 2.2). By observing the program, it can be determined that the variable `z` is not present in the data set `third` and the variable `w` is not

present in the data set **fourth**, whereas the variables **x** and **y** are common to both data sets. The data set **fifth** is formed using the data step

```
data fifth;
  set third fourth;
```

The data set **fifth** is formed by concatenating the observations in the two data sets **third** and **fourth** and so will contain *six* observations and *four* variables **w**, **x**, **y**, and **z**. In the simplest use of the set statement illustrated here, SAS reads observations from the first data set in the list, **third**, transfers data values to the PDV, and the writes them sequentially to the new data set **fifth**. For example, the PDV following reading the first observation is

w	x	y	z	_N_	_ERROR_
21	102	3	.	1	0

Although, only the variables **w**, **x**, and **y** are in data set **third**, SAS has detected the presence of the variable **z** in the data step during the compiling stage. Thus, a slot for **z** is created in the PDV and a missing value is inserted at initialization. When the observation is written to the output data set **fifth**, it will contain the values for the *four* variables **w**, **x**, **y**, and **z** as given above.

```
2  data third ;
3  input w 1-2 x 3-5 y 6;
4  datalines;

NOTE: The data set WORK.THIRD has 3 observations and 3 variables.
NOTE: DATA statement used (Total process time):

8  ;
9  run;
10 data fourth;
11 input x y z;
12 datalines;

NOTE: The data set WORK.FOURTH has 3 observations and 3 variables.
NOTE: DATA statement used (Total process time):

16 ;
17 run;
18 data fifth;
19 set third fourth;
20 run;

NOTE: There were 3 observations read from the data set WORK.THIRD.
NOTE: There were 3 observations read from the data set WORK.FOURTH.
NOTE: The data set WORK.FIFTH has 6 observations and 4 variables.

21 proc print;
22 title 'Combining SAS data sets end-to-end ';
23 run;

NOTE: There were 6 observations read from the data set WORK.FIFTH.
NOTE: PROCEDURE PRINT used (Total process time):
```

Fig. 2.2. SAS Example B1: Log

Once the data in the first data set listed is exhausted, SAS begins reading data from the second data set **fourth** and transfers data values to the PDV. The PDV following reading the first observation from the data set **fourth** is

w	x	y	z	_N_	_ERROR_
.	14	5	7862	1	0

Again, an observation containing values for the variables **w**, **x**, **y**, and **z** is written to the output data set **fifth**. This process continues until data set **fourth** reaches end-of-file. Then the data step comes to an end and SAS closes the output data set **fifth** and exits. The number of observations in the new data set is the total number of observations in the two input data sets, and the order of appearance is all observations from the first data set followed by all observations from the second data set, with missing values set appropriately for **z** and **w**, respectively. The output from **proc print** (shown in Fig. 2.3) displays a listing of the data set **fifth**.

Combining SAS data sets end-to-end					1
Obs	w	x	y	z	
1	21	102	3	.	
2	31	203	4	.	
3	41	304	5	.	
4	.	14	5	7862	
5	.	15	6	6517	
6	.	16	7	8173	

Fig. 2.3. SAS Example B1: Output

The SET Statement

The general form of the **set** statement is

```
SET <SAS-data-set(s) <(data-set-option(s))> > <options> ;
```

where *data-set-options* are those options that may be specified in parentheses after a SAS data set name, whether it is an input data set (as in a **set** statement) or an output data set (as in an **input** statement). More commonly used options such as **firstobs=**, **obs=**, or **where=** specify observations; those such as **drop=**, **keep=**, and **rename=** have variable names as arguments. When a set statement is used, it is more efficient to use an option to access only those variables required:

```
data fifth;
set third(keep=x y) fourth(drop=z);
run;
```

This will result in a SAS data set named **fifth** without any missing values:

Obs	x	y
1	102	3
2	203	4
3	304	5
4	14	5
5	15	6
6	16	7

If the variable **z** in the SAS data set **fourth** is renamed to be **w**, it will also result in a SAS data set with no missing values (albeit one different from the above):

```
data fourth;
set third fourth(rename=(z=w));
run;
```

This would be an option if variables measuring the same quantity have been assigned different names in the two data sets. By renaming **z** to be **w**, a variable that already exists in the data set **third**, the user is in fact recognizing this fact. The resulting data set is

Obs	w	x	y
1	21	102	3
2	31	203	4
3	41	304	5
4	7862	14	5
5	6517	15	6
6	8173	16	7

In the SAS Example A2 program (see Fig. 1.5), the data set option **where=** could have been used to select the required subset of observations; thus,

```
data second;
set first(where=(age<35 & state='IA'));
run;
```

There are several options that are unique to the **set** statement; among them are those that enable accessing observations nonsequentially according to a value given in the **key=** option or according to the observation number in the **point=** option.

Programming statements other than the *subsetting if*, such as assignment statements, may be used following the **set** statement, just as one would use following an **input** statement. In particular, one could use the **output** statement to create multiple observations in the output data set from a single observation in the input data set, similar to its use in SAS Example A7 (see

Fig. 1.18). The use of a `by` following the `set` statement allows interleaving of observations in several data sets. The observations in the output data set are arranged by the values of the `by` variable(s), in the order of the data sets in which they occur. Consider the two data sets AAA and BBB containing information for identical subjects:

Data set AAA		Data set BBB	
Id	Height	Id	Weight
111	65	111	145
222	70	222	156
333	58	333	148
444	71	444	166
555	69	555	175
777	70	666	136

The following SAS data step results in the formation of an interleaved SAS data set:

```
data CCC;
  set AAA BBB;
  by id;
run;
```

The resulting data set CCC (a listing is shown below) has 12 observations, which is the total number of observations from both data sets. The new data set contains all variables from both data sets. The values of variables found in one data set but not in the other are set to a missing value, and the observations are arranged in the order of the values of the variable `id`. In particular, note that the observation with `id` equal to 666 occurs before that with the `id` equal to 777 in the output data set, although the second observation came from the data set AAA listed first in the `set` statement. Note that observations in each of the original data sets were already arranged in the increasing order of the values of `id`. Thus, it is required for interleaving to ensure that the observations are sorted or grouped in each input data set by the variable or variables that are in the `by` statement.

id	height	weight
111	65	.
111	.	145
222	70	.
222	.	156
333	58	.
333	.	148
444	71	.
444	.	166
555	69	.
555	.	175
666	.	136
777	70	.

Instead of a `set` statement, a `merge` statement may be used to combine these two data sets:

```
data CCC;
  merge AAA BBB;
  by id;
run;
```

A listing of the resulting data set is

	Obs	id	height	weight
	1	111	65	145
	2	222	70	156
	3	333	58	148
	4	444	71	166
	5	555	69	175
	6	666	.	136
	7	777	70	.

Note that missing values are generated for the variables `height` and `weight` for those observations with no common `id` values in both data sets.

2.1.3 Saving and retrieving permanent SAS data sets

In SAS examples discussed so far in this chapter, raw data, input either in-stream or from text files, were used to create temporary SAS data sets. As discussed in Section 1.2, SAS data sets contain not only the rectangular array of data but also other information such as variable attributes. In practice, the creation of a SAS data set requires substantial effort so that a user may want to save it permanently for future analysis using SAS procedures for performing different statistical applications. The availability of a carefully constructed permanent SAS data set allows the user to bypass the data set creation step at least for the duration of a research project. In addition, SAS data sets have become a convenient vehicle for transfer of large data sets to other users.

Two SAS examples are used in this subsection to illustrate how to use raw data to create a permanent data set and how to retrieve data for analysis from a previously saved SAS data set. The concept of a SAS *library* is easily understood in the context of running SAS programs under the Windows environment. Recall that the complete path name of a file was used with the `filename` statement to associate a fileref with the physical name and location of a file. Similarly, the `libname` statement associates the physical name and location of an external folder (directory) with a *libref*, which is an alias for the complete path name of the folder (directory). The following statement assigns the libref `mylib` to the folder named `projectA`:

```
libname mylib 'C:\Documents and Settings\...\projectA\';
```

To save a SAS data set in a folder given in a libref as above, the user must specify a *two-level* SAS data set name, where the first level is the libref and the second level is the actual data set name. A two-level SAS data set name, in general, is a name that contain two parts separated by a period of the form **libref.membername** and is used to refer to *members* of a library libref. The *membername* is the name of a SAS data set when the members stored in the library are SAS data sets. Under the Windows operating system, a library is synonymous with a folder (or a directory). Thus, SAS data sets can be saved in a folder directly by executing statements in SAS programs giving two-level names to the data sets to be saved.

For example, **mylib.survey** refers to a SAS data set named **survey** to be saved in the above folder **projectA**. The libref defined in a SAS program is available for use only within the current SAS program. Many SAS data sets may be saved in the same folder (as members of the library) by using the libref **mylib** as the first-level name as many times as needed in the same program. A different name may be used as a libref to associate the same library in another SAS program, thus allowing the user to access previously stored members or add new members to the library.

SAS Example B2

```
libname mylib1 'G:\stat479\Class\';
data mylib1.first;
input x1-x5;
datalines;
1 2 3 4 5
2 3 4 5 6
6 5 4 3 2
1 2 1 2 1
7 2 55 5 5
;
```

Fig. 2.4. SAS Example B2: Program

The SAS Example B2 program (see Fig. 2.4) is a simple example illustrating the use of the **libname** statement and two-level names to create and access permanent SAS data sets. Instream raw data lines are used to create a SAS data set using the two-level name **mylib1.first**. The first part of the two-level name **mylib1** refers to a folder in a disk mounted on a zip drive. Thus, the SAS data set named **first** created in the data step is saved as a permanent file in the specified folder. Thus, the data set **first** will be a member of this library. The SAS log reproduced in Fig. 2.5 indicates this fact by listing the two-level name **MYLIB1.FIRST** and identifying the name of the folder as **G:\stat479\Class**. The actual physical name of the file saved is

```

2  libname mylib1 'G:\stat479\Class\';
NOTE: Libref MYLIB1 was successfully assigned as follows:
      Engine:          V9
      Physical Name:   G:\stat479\Class
3  data mylib1.first;
4  input x1-x5;
5  datalines;

NOTE: The data set MYLIB1.FIRST has 5 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time          0.03 seconds
      cpu time           0.01 seconds

11 ;
12 run;

```

Fig. 2.5. SAS Example B2: Log page

`first.sas7bdat`, as can be verified by manually obtaining a listing of the `Class` folder (see Fig. 2.6). Obviously, if a single-level name, say `first`, was used instead in the data statement, the SAS data set would have been temporarily saved in the `WORK` folder (and the SAS data set thus created referred to as `WORK.FIRST` in the log page).

Name	Size	Type	Date Modified
cities2.sas7bdat	97 KB	SAS Data Set	6/28/2006 11:59 AM
cities.sas7bdat	61 KB	SAS Data Set	6/28/2006 11:23 AM
first.sas7bdat	5 KB	SAS Data Set	9/5/2007 11:41 AM
fuel.data	2 KB	DATA File	11/3/2004 11:45 AM
second.sas7bdat	5 KB	SAS Data Set	12/19/2007 4:23 PM
zany.data	1 KB	DATA File	9/14/2004 9:21 AM

Fig. 2.6. Screen shot of `Class` folder listing

SAS Example B3

By including a `libname` statement of the form shown in the SAS program shown in Fig. 2.4 (possibly with a different libref, but the same physical path name of the folder), one or more SAS data sets stored permanently in a library can be accessed for further processing in another SAS program to be executed subsequently.

In the SAS Example B3 program (see Fig. 2.7) the SAS data set `first` is accessed from the library for processing using this method. The following `libname` statement in this program associates the libref `mydef1` with the same library where the data set `first` was saved when the SAS program in Fig. 2.4 was executed:

```
libname mydef1 'G:\stat479\Class\';
```

```

libname mydef1 'G:\stat479\Class\';

proc print data=mydef1.first; run;

proc means data=mydef1.first; run;

data mydef1.second;
input y1-y3;
datalines;
31 34 38
43 45 47
10 11 12
908 97 96
;
run;
proc contents data=mydef1.first; run;

proc datasets library=mydef1 memtype=data;
  contents data=first directory details;
run;

```

Fig. 2.7. SAS Example B3: Program

This allows the two-level name `mydef1.first` to be used as shorthand for accessing the SAS data set `first` from the library to be analyzed using the SAS procedure `proc print` by naming it in the `data=` option. The listing resulting from this statement is shown on page 1 of the output produced by the SAS Example B3 program, displayed in Fig. 2.8.

The SAS System						1
	Obs	x1	x2	x3	x4	x5
	1	1	2	3	4	5
	2	2	3	4	5	6
	3	6	5	4	3	2
	4	1	2	1	2	1
	5	7	2	55	5	5
The SAS System						2
The MEANS Procedure						
Variable	N	Mean	Std Dev	Minimum	Maximum	
x1	5	3.4000000	2.8809721	1.0000000	7.0000000	
x2	5	2.8000000	1.3038405	2.0000000	5.0000000	
x3	5	13.4000000	23.2873356	1.0000000	55.0000000	
x4	5	3.8000000	1.3038405	2.0000000	5.0000000	
x5	5	3.8000000	2.1679483	1.0000000	6.0000000	

Fig. 2.8. SAS Example B3: Output pages 1 and 2

The statement `proc means data=mydef1.first;` produces the statistical analysis shown on page 2 of the output from the SAS Example B3 program displayed in Fig. 2.8. Again, `proc means` accesses the SAS data set `first` from the same library and computes the default statistics for all variables in the data set as shown on page 2. There is nothing to preclude the user from adding new SAS data sets to the same library, in the same program or in separate SAS programs. The data step shown in Fig. 2.7 reads instream data using list input as usual and creates the SAS data set named `second` and then saves it permanently in the library identified by the libref `mydef1`.

The SAS System				3
The CONTENTS Procedure				
Data Set Name	MYDEF1.FIRST	Observations		5
Member Type	DATA	Variables		5
Engine	V9	Indexes		0
Created	Wed, Sep 05, 2007 11:41:54 AM	Observation Length		40
Last Modified	Wed, Sep 05, 2007 11:41:54 AM	Deleted Observations		0
Protection		Compressed		NO
Data Set Type		Sorted		NO
Label				
Data Representation	WINDOWS_32			
Encoding	wlatin1 Western (Windows)			
Engine/Host Dependent Information				
Data Set Page Size	4096			
Number of Data Set Pages	1			
First Data Page	1			
Max Obs per Page	101			
Obs in First Data Page	5			
Number of Data Set Repairs	0			
File Name	G:\stat479\Class\first.sas7bdat			
Release Created	9.0101M3			
Host Created	XP_PRO			
Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	
1	x1	Num	8	
2	x2	Num	8	
3	x3	Num	8	
4	x4	Num	8	
5	x5	Num	8	

Fig. 2.9. SAS Example B3: Output page 3

The SAS procedure `contents` enables the user to examine the contents of SAS data sets stored in a library. The output from the first use of `proc contents` in the SAS Example B3 program is displayed in Fig. 2.9. This contains general technical information about the file stored, such as the length of an observation and the version of SAS used to create the data set as well

as more specific information such as the number of variables and the number of observations. For a user accessing a SAS data set from another source, the more useful information appears at the bottom of the page. This is an alphabetic list of the variables and their attributes. Note that if the data analyst who originally created this data set has defined such information as formats and labels for the variables in the data step, they would appear here.

A more generally applicable SAS procedure named **datasets** is also available for examining the contents of an existing SAS library. An illustration of the use of this procedure is shown in Fig. 2.7. Here, the procedure statement option **library=** names the library to be examined and the **memtype=** option names the member type of interest. The **data=** option in the **contents** statement can be set to the name of a specific member to be examined or to the SAS keyword **_ALL_** to indicate that all data sets are to be examined. The keyword option **directory** requests that a list of the SAS data sets in the SAS library be printed and the **details** option requests that information contained in the data sets, such as the number of observations and number of variables, be included in the output. The printed output from the above **proc datasets** step is identical to the output from **proc contents** shown in Fig. 2.9.

2.1.4 User-defined informats and formats

Before discussing the use of the **format** procedure for creating user-defined informats and formats, a review of these two variable attributes is informative. Several simple informats such as **\$10.** or **5.2** and more complex informats such as **dollar10.2** or **monyy5.** were used in several examples in Chapter 1. **Informats** determine how raw data values are read and converted to a number or a character string to be stored in memory locations. An informat contains information of the type of data (character or numeric) to be read, the length it occupies in the data field, how to handle leading, trailing, or embedded blanks and zeros, where to place the decimal point, and so forth. For example, the informat **ddmmyy8.0** converts the date value 19/10/07 entered in a data line into the binary number 17458 to be stored as a value of a SAS variable. Similarly, **formats** convert data values from internal form into a form the user wants them to appear in printed output. For example, the format **dollar15.2** prints the value of **cost=2317438.3921**, which is (say) the result of the product of the values of **quantity=2346.678** and **price=987.54**, as \$2,317,438.39.

SAS system contains a large number of predefined informats and formats to handle many types of data conversion. However, it is not possible to provide informats or formats for every conceivable need; for example, an informat might be needed to convert the character strings 'YES' and 'NO' to be stored as the numeric values one or zero, respectively; or a numeric value stored internally as a one or a two to indicate gender will be best converted to be output as character strings 'Female' or 'Male', respectively. **Proc format** is a SAS procedure that allows the user to define informats or formats to do these

kinds of specialized conversion. In this section, the emphasis will be on the use of `proc format` for creating output formats, although the importance of user-defined informats cannot be overstated. In practical terms, the primary use of user-constructed informats is for data validation and some examples of this type of application appear below. The general structure of a `proc format` step (with three procedure information statements to be illustrated below) is

```
PROC FORMAT <option(s)>;
  INVALUE <$>name <(informat-option(s))> value-range-set(s);
  PICTURE name <(format-option(s))> value-range-set-1
                                <(picture-1-option(s))>
                                <...value-range-set-n <(picture-n-option(s))> >;
  VALUE <$>name <(format-option(s))>value-range-set(s);
```

In the above description, the phrase *value-range-set* refers to an assignment type specification that defines a one-to-one or a many-to-one relationship between value or values to be converted to another value. The specification and action of a *value-range-set* depends on the context of its usage.

In the case of an `invalue` statement, *value-range-set* is of the form

```
value or range = informatted-value|[existing-informat]
```

where *value* is a value such as 1 or 'NB', *range* is a list of values usually specified in the form 100-999 or 'A'-'Z'. The words *low* and *high* may be used to define the end points of any range (numeric or character), implying that the specified range covers the entire range of values below the upper end point or above the lower end point, respectively. For example, the range 100-*high* covers every value greater than 100. The *informatted-value* (on the right-hand side of the equal sign) specifies the internal value that the raw data value that is equal to the value (or in the range of values) on the left is to be converted.

In the case of a `value` statement, the *value-range-set* is of the form

```
value or range = 'formatted-value'|[existing-format].
```

The definition of *value* and *range* is the same as for the `invalue` statement. The '*formatted-value*' specifies a character string to which the value (or the range of values) that appears on the left side of the equal sign is to be converted for printing. The '*formatted-value*' is a character string, regardless of whether the format created is a character or numeric format.

A discussion of several possible options on the *value* and *invalue* statements are omitted here but can be found under the description of `proc format`. For example, the option `fuzz=` allows the user to specify a fuzz factor for matching values to a range. If a value does not exactly match or falls in a range but comes within the fuzz factor, then the format or the informat will consider it to be a match or in the range. This facility is useful especially when the raw data contains fractions that need to be rounded up or down to be exactly in a prespecified range. For example, the value 99.9 may be considered

in the range 100-200 if a fuzz factor of .1 has been specified (`fuzz=.1`) and values below 100 are not considered to be in the conversion range.

SAS Example B4

```
proc format;
    invalue $st 'IA'='Iowa'
               'NB'='Nebraska';
run;
data first ;
length state $ 12;
infile 'C:\Documents and Settings\...\wages.txt';
input (income tax age state)(@4 2*5.2 2. $st2.);
run;
proc print noobs;
format income tax dollar8.2 state $12. ;
var income tax age state;
title 'SAS Listing of Tax data';
run;
```

Fig. 2.10. SAS Example B4: Program

In the SAS Example B4 program displayed in Fig. 2.10, the two-character state codes used in the raw data set of SAS Example A1 (see Fig. 1.1) are converted to values that are longer character strings identifying the name of the respective state. Since there are several SAS functions (e.g., `stname1()`) available for state name conversions, this informat (named `$st`) is created only as an illustration. In the `proc format` step, an `invalue` statement is used to define the required conversion. Note that only values expected to be in the data for the character variable `state` are used in this definition. If other state values are expected, then they must be included in the format definition. Since character type variables are assigned lengths of 2 bytes by default, a `length` statement (that must appear before the `input` statement) specifies the length of the `state` variable to be 12 bytes. Thus, a format with a width of at least 12 positions is needed to print values of the `state` variable. As seen in the SAS program, the format `$12.` is associated with the `state` variable and the resulting output is shown in Fig. 2.11.

If the only state codes allowed in the data set are 'NB' and 'IA', the `invalue` statement may be modified to flag any other state code used as an error as follows:

```
invalue $st 'IA'='Iowa' 'NB'='Nebraska' other='Invalid St.';
```

In the above, the word `other` is a SAS keyword that will match any value that is not the strings 'NB' or 'IA'. Thus, if this informat is used to input values for a character variable with values other than 'NB' or 'IA', the respective observations will contain the string 'Invalid St.' as the value of that variable.

SAS Listing of Tax data				1
income	tax	age	state	
\$546.75	\$34.65	35	Iowa	
\$765.48	\$89.56	45	Iowa	
\$578.65	\$59.54	31	Iowa	
\$786.78	\$57.65	41	Nebraska	
\$567.51	\$126.85	32	Iowa	
\$785.87	\$67.85	28	Nebraska	
\$985.65	\$75.65	43	Nebraska	
\$745.63	\$78.95	25	Iowa	
\$345.67	\$25.68	23	Nebraska	
\$567.34	\$89.75	34	Nebraska	
\$651.12	\$50.45	45	Iowa	
\$650.75	\$65.45	29	Nebraska	
\$595.65	\$45.68	34	Iowa	
\$678.56	\$91.27	28	Nebraska	
\$685.96	\$67.51	38	Iowa	
\$825.75	\$56.25	27	Iowa	

Fig. 2.11. SAS Example B4: Output

This is an example of the use of an informat for data validation, an important step in data analysis.

It is important to note that user-defined informats read only **character** values, although these can be converted to either character or numeric values. In the above example, if the state FIPS codes were input as numbers (19 for IA and 31 for NB), they must still be accessed as character data by the informat. So the appropriate `invalue` statement is

```
invalue $st '19'='Iowa' '31'='Nebraska';
```

If the expansion of the state code was required only for the printed output, then it would have been sufficient to create a format (as opposed to an informat) for this purpose. The above program is modified as shown in Fig. 2.12 (the output from this SAS program is not shown).

```
proc format;
  value $st 'IA'='Iowa'
            'NB'='Nebraska';
run;
data first ;
infile 'C:\Documents and Settings\...\wages.txt';
input (income tax age state)(@4 2*5.2 2. $2.);
run;
proc print noobs;
format income tax dollar8.2 state $st10. ;
var income tax age state;
title 'SAS Listing of Tax data';
run;
```

Fig. 2.12. SAS Example B4: Modified program

A **value** statement is used to define a format for printing values of the variable **state**. Note that the new format **\$st** is used in a **format** statement to specify the values of the variable **state**. Note carefully that the values to be stored in **state** were read using the informat **\$2.** and hence will be one of the strings 'IA' or 'NB'. The conversion takes place when they are output using the format **\$st10.**, where these values will be printed as 'Iowa' and 'Nebraska', respectively, in using 10 print positions aligned to the left.

Note the difference between the **format** procedure and the **format** statement carefully. As in the above example, **proc format** is used to create user-defined formats or informats. The **format** (or the **informat**) statement associates an existing format (or informat) with one or more variables. Either standard SAS or user-defined formats or informats can be associated with variables this way. For example, the statement

```
format income tax dollar8.2 state $st10.;
```

associates the SAS format **dollar8.2** with the variables **income** and **tax**, whereas the user-defined format **\$st10.** is associated with the variable **state**. **Proc format** stores user-defined informats and formats as entries in *SAS catalogs* (specially structured files), either temporarily in the WORK library or permanently in a user-specified library.

Finally, the following example illustrates how an existing SAS informat or a format can be used as an informatted or a formatted value in **value-range-set** definitions in **invalue** or **value** statements, respectively. Recall that the definitions of **value-range-sets** in these two statements were

```
value or range = informatted-value|[existing-informat]

value or range = 'formatted-value'|[existing-format].
```

Instead of an informatted or a formatted value, the user can specify an existing SAS informat or a format placed inside box brackets that will be used for the conversion of the value or the range on the left hand side of the **value-range-set** definition.

Example 2.1.2

```
proc format;
  invalue ff 0-high=[4.2]
              -1 = .;
run;

data ex212;
input a 2.0 b ff4.2 ;
datalines;
10 205
20 216
```

```

30 237
40 257
50 -1
60 469
;
run;

```

The user-defined numeric informat (named **ff**) converts all positive data values using the SAS informat 4.2. When the data value is -1 , it is converted to the SAS missing value for a numeric variable i.e., a period . A scenario for the need to use such an informat may arise if the raw data set has been prepared where a -1 has been entered instead of using SAS missing values or spaces to indicate missing data values where the actual data values are all positive numbers. The output data set is

Obs	a	b
1	10	2.05
2	20	2.16
3	30	2.37
4	40	2.57
5	50	.
6	60	4.69

2.1.5 Creating SAS data sets in procedure steps

Several SAS procedures used for statistical analysis have the capability to let the user specify which statistics, calculated by the procedure, are to be saved in newly created SAS data sets. In some procedures, these data sets are organized in special structures that allow them to be read by another SAS procedures for further analysis by specifying the **type=** attribute of the data set. For example, **proc corr** creates a data set with the attribute **type=corr** containing a correlation matrix, that can be directly input to a procedure such as **proc reg** as an input data set. If the required analysis performed by **proc reg** is solely based on the correlation matrix then much of the overhead spent on recomputing the correlation matrix can be avoided.

In this subsection, the discussion is limited to a description of the use of the **output** statement in several SAS procedures that compute an extensive number of statistics for variable values. In most of these procedure steps, a **class** statement specifies classification variables in the data set that are discrete-valued variables that identify groups, classes, or categories of observations in the data set. They may be numeric or character-valued and may be observed ordinal- or nominal-valued variables or user-constructed variables. In practice, continuous-valued variables may be used to define new grouping variables that can then be used in **class** statements. An example would be

the creation of a variable defining income groups with values, say, 1, 2, and 3, or ‘Low’, ‘Medium’, and ‘High’, using the values of the continuous-valued variable `income` to form the groups. A `var` statement (i.e., the variables statement) identifies the analysis variables (that must all be of numeric type). The statistics are computed on the values of analysis variables for subsets of observations defined by the classification variables.

In the SAS Example B5 program, `proc means` is used to introduce the basic use of the output statement. A simplified general form of the `output` statement used in this example is

```
OUTPUT <OUT=SAS-data-set> <output-statistic-specification(s)>;
```

where an *output-statistic-specification* is of the general form

```
statistic-keyword<(variable-list)>=<name(s)>
```

where *statistic-keyword* specifies the statistic to be calculated and is stored as a value of a variable in the output data set. Some of available statistic keywords are `n`, `mean`, `median`, `var`, `cv`, `std`, `stderr`, `max`, `min`, `range`, `q1`, `q3`, `qrange`, `p1`, `p5`, `p10`, `p90`, `p95`, `p99`, `t`, and `probt`. The optional *variable-list* specifies the names of one or more analysis variables on whose values the specified statistic is to be computed. If this list is omitted, the specified statistic is computed for all the analysis variables. The optional *name(s)* specifies one or more names for the variables in the output data set that will contain the analysis variable statistics in the same sequence that the analysis variables are listed in the `var` statement. The first name contains the statistic for the first analysis variable; the second name contains the statistic for the second analysis variable; and so on. If the names are omitted, the analysis variable names are used to name the variables in the output data set.

SAS Example B5

The SAS Example B5 program (see Fig. 2.13) illustrates the use of `proc means` to calculate and print statistics for an input data set named `biology` and, in addition, save the statistics in a new SAS data set created in the `proc` step. The simple data set to be analyzed includes a numeric variable `year` (indicating class in college) and a character variable `sex` that will be used as classification variables and two numeric analysis variables `height` and `weight`. Ordinarily, `proc means` produces printed output of five default statistics (`n` (sample size), mean, standard deviation, minimum, and maximum) calculated for every variable in the `var` statement list, for subsets of observations formed by all combinations of the levels of the `class` variables. The option `maxdec=3` used on the `proc` statement limits the number of decimal places output when printing all calculated statistics. Page 1 of the SAS output (see Fig. 2.14) displays the printed output in its standard format. As described above, the five default statistics are computed for the variables `height` and `weight` for

```

data biology;
input id sex $ age year height weight;
datalines;
7389 M 24 4 69.2 132.5
3945 F 19 2 58.5 112.0
4721 F 20 2 65.3 98.6
1835 F 24 4 62.8 102.5
9541 M 21 3 72.5 152.3
2957 M 22 3 67.3 145.8
2158 F 21 2 59.8 104.5
4296 F 25 3 62.5 132.5
4824 M 23 4 74.5 184.4
5736 M 22 3 69.1 149.5
8765 F 19 1 67.3 130.5
5734 F 18 1 64.3 110.2
4529 F 19 2 68.3 127.4
8341 F 20 3 66.5 132.6
4672 M 21 3 72.2 150.7
4823 M 22 4 68.8 128.5
5639 M 21 3 67.6 133.6
6547 M 24 2 69.5 155.4
8472 M 21 2 76.5 205.1
6327 M 20 1 70.2 135.4
8472 F 20 4 66.8 142.6
4875 M 20 1 74.2 160.4
;
run;

proc means data=biology fw=8 maxdec=3;
class year sex;
var height weight;
output out=stats mean=av_ht av_wt stderr=se_ht se_wt;
run;

proc print data=stats;
title 'Biology Class Data Set: Output Statement';
run;

```

Fig. 2.13. SAS Example B5: Program

groups observations defined by the levels 'F' and 'M', respectively, of the **sex** variable within each value 1, 2, 3, or 4, of the **year** variable, respectively.

The *output-statistic-specifications* used in the **output** statement has the basic form of **statistic=names**. They are **mean=av_ht av_wt** and **stderr=se_ht se_wt**. Since the **var** statement used in the **proc** step is **var height weight;**, the above specifications request that the means and standard errors of the variables **height** and **weight** are to be computed and stored in the new variables **av_ht**, **av_wt**, **se_ht**, and **se_wt**, respectively. Page 2 of the SAS output (see Fig. 2.15) displays a listing of the SAS data set named **stats** produced in the **proc means** step. It can be observed that there are 15 observations displaying values for the variables **year**, **sex**, **_TYPE_**, **_FREQ_**, **av_ht**, **av_wt**, **se_ht**, and **se_wt**. The value of the **_TYPE_** variable (0, 1, 2, or 3) indicates which combinations of the class variables are used to define the subgroups of observations used for computing the statistics. For example, there is exactly one observation (Observation 1) with the value **_TYPE_=0**. In

Biology Class Data Set: Output Statement								1
The MEANS Procedure								
year	sex	N	Variable	N	Mean	Std Dev	Minimum	Maximum
1	F	2	height	2	65.800	2.121	64.300	67.300
			weight	2	120.350	14.354	110.200	130.500
	M	2	height	2	72.200	2.828	70.200	74.200
			weight	2	147.900	17.678	135.400	160.400
2	F	4	height	4	62.975	4.614	58.500	68.300
			weight	4	110.625	12.455	98.600	127.400
	M	2	height	2	73.000	4.950	69.500	76.500
			weight	2	180.250	35.143	155.400	205.100
3	F	2	height	2	64.500	2.828	62.500	66.500
			weight	2	132.550	0.071	132.500	132.600
	M	5	height	5	69.740	2.481	67.300	72.500
			weight	5	146.380	7.535	133.600	152.300
4	F	2	height	2	64.800	2.828	62.800	66.800
			weight	2	122.550	28.355	102.500	142.600
	M	3	height	3	70.833	3.182	68.800	74.500
			weight	3	148.467	31.183	128.500	184.400

Fig. 2.14. SAS Example B5: Output page 1

this observation, the variables `year` and `sex` are set to respective missing values, indicating that both of these variables are ignored in determining the sample used to compute the statistics shown for this observation; that is, the subgroup for their computation is the entire data set as evidenced by the value of `_FREQ_=22`.

Similarly, for `_TYPE_=1`, there are two subgroups formed for each of the values of the `sex` variable ignoring the `year` variable. Note carefully that `sex` variable appears rightmost in the variable list in the `class` statement; hence, its levels form `_TYPE_=1` groups. Observations 2 and 3 list statistics computed based on these groups of observations and note sample sizes given by `_FREQ_=10` and `_FREQ_=12`, respectively.

There are four observations with `_TYPE_=2`, and these statistics are based on the groups of observations that correspond to each level of `year` ignoring the levels of `sex`. Observations with `_TYPE_=3` correspond to subgroups defined by all combinations of levels of `year` and the levels of `sex`. Thus, the complete set is formed by $1+2+4+8 = 15$; thus, 15 observations are included in `stats`.

Other forms of the *output-statistic-specifications* used in the `output` statement can be used to alter the appearance of the SAS data set created. Several procedure information statements and proc statement options are available

Biology Class Data Set: Output Statement								2
Obs	year	sex	_TYPE_	_FREQ_	av_ht	av_wt	se_ht	se_wt
1	.		0	22	67.8955	137.591	0.97773	5.4643
2	.	F	1	10	64.2100	119.340	1.03489	4.8887
3	.	M	1	12	70.9667	152.800	0.85390	6.4766
4	1		2	4	69.0000	134.125	2.11069	10.3181
5	2		2	6	66.3167	133.833	2.72255	16.4964
6	3		2	7	68.2429	142.429	1.30783	3.4515
7	4		2	5	68.4200	138.100	1.89642	13.3320
8	1	F	3	2	65.8000	120.350	1.50000	10.1500
9	1	M	3	2	72.2000	147.900	2.00000	12.5000
10	2	F	3	4	62.9750	110.625	2.30701	6.2277
11	2	M	3	2	73.0000	180.250	3.50000	24.8500
12	3	F	3	2	64.5000	132.550	2.00000	0.0500
13	3	M	3	5	69.7400	146.380	1.10932	3.3698
14	4	F	3	2	64.8000	122.550	2.00000	20.0500
15	4	M	3	3	70.8333	148.467	1.83697	18.0037

Fig. 2.15. SAS Example B5: Output page 2

for controlling the contents of this data set. The following examples of the **ways** and **types** statements illustrate some of these choices. These two statements allow the user to select the set of observations to be included in the output data set as defined by the **_TYPE_** variable discussed earlier. The **ways** statement uses integers to indicate number of class variables to form the combinations; for example, one may specify 2 to request that subgroups are to be formed by combining all possible pairs of class variables in the class variable list. The **types** statement, on the other hand, allows the user to specify class variables and how they are to be combined directly.

Including the statement **ways 1;** in the proc step in the above example produces the output data set shown in Fig. 2.16. This requests that subgroups are to be defined by the levels of the class variables taken one at a time. Here, two sets of statistics are produced for the two levels of **sex** and four sets for the four levels of **year**. The printed output (not shown) is similarly structured; two tables of statistics are produced for each class variable separately.

Biology Class Data Set: Output Statement								2
Obs	year	sex	_TYPE_	_FREQ_	av_ht	av_wt	se_ht	se_wt
1	.	F	1	10	64.2100	119.340	1.03489	4.8887
2	.	M	1	12	70.9667	152.800	0.85390	6.4766
3	1		2	4	69.0000	134.125	2.11069	10.3181
4	2		2	6	66.3167	133.833	2.72255	16.4964
5	3		2	7	68.2429	142.429	1.30783	3.4515
6	4		2	5	68.4200	138.100	1.89642	13.3320

Fig. 2.16. SAS Example B5: Result using the WAYS statement

Biology Class Data Set: Output Statement									2
Obs	year	sex	_TYPE_	_FREQ_	av_ht	av_wt	se_ht	se_wt	
1	1	F	3	2	65.8000	120.350	1.50000	10.1500	
2	1	M	3	2	72.2000	147.900	2.00000	12.5000	
3	2	F	3	4	62.9750	110.625	2.30701	6.2277	
4	2	M	3	2	73.0000	180.250	3.50000	24.8500	
5	3	F	3	2	64.5000	132.550	2.00000	0.0500	
6	3	M	3	5	69.7400	146.380	1.10932	3.3698	
7	4	F	3	2	64.8000	122.550	2.00000	20.0500	
8	4	M	3	3	70.8333	148.467	1.83697	18.0037	

Fig. 2.17. SAS Example B5: Result from using TYPES statement

Other possibilities in this example are `ways 0`; when no subgroups are formed, meaning statistics are computed for the entire data set, and `ways 2`; when subgroups are formed for all eight combinations of the two class variables.

Including the statement `types year sex;` in the proc step produces the same data set shown in Fig. 2.16 and the corresponding printed output (not shown). If, instead, `types year;` is used, then only those statistics for the subgroups defined by the four levels of `year` will be calculated. The statement `types year*sex;` produces statistics for subgroups formed for the eight combinations of the two class variables `year` and `sex` as shown in Fig. 2.17.

2.2 SAS Procedures for Computing Statistics

The data set shown in Table B.1 of Appendix B appeared in Weisberg (1985) and was extracted from the *American Almanac* and the *World Almanac* for 1974. It lists the values of fuel consumption for each of the 48 contiguous states, in addition to several other measured variables. This data set is used to illustrate several SAS procedures that are classified as Base SAS procedures. As a prelude to the use of several SAS procedures for analysis, a SAS data set that contains user-generated labels, formats, grouping variables (ordinal or nominal variables with values that identify groups of observations belonging to different classes or strata), etc. is created and stored in a library. This data set is then accessed repeatedly in several SAS proc steps.

SAS Example B6

The SAS data set `fueldata` is created in the SAS Example B6 program shown in Fig. 2.18. The following actions are taken in the data step of this program. The data are input from a text file using an `infile` statement. The SAS data set is saved in a folder using the two-level name `mylib.fueldata` to be accessed in other SAS programs later. Mnemonic variable names are used, but `label` statements are included to provide more descriptive labeling as necessary. In the same data step, five new variables are created as follows:

```

libname mylib 'C:\Documents and Settings\...\stat479\';
data mylib.fueldd;
filename fueldd 'C:\Documents and Settings\...\fuel.txt';
infile fueldd;
input (st pop tax numlic income roads fuelc)
      ($2. 5. 2.1 5. 4.3 5.3 5.);

label pop='Population(in thousands)'
      tax='Motor Fuel Tax Rate(in cents/gallon)'
      numlic='No. of Licensed Drivers'
      income='Per Capita Income(in thsnds.)'
      roads='Miles of Primary Highways(in thsnds.)' ;

percent=100*numlic/pop;

fuel=1000*fuelc/pop;

if income<=3.8 then incomgrp=1;
else if 3.8<income<=4.4 then incomgrp=2;
else incomgrp=3;

if tax<8.0 then taxgrp='Low ';
else taxgrp='High';

label percent='% of Population with Driving Licenses'
      fuel='Fuel Consumption (in gallons/person)'
      incomgrp='Per capita Income'
      taxgrp='Fuel Tax'
      state='State' ;

format percent 4.1 fuel 7. ;

state=stname(st);

drop fuelc st;
run;

proc print label;
title 'Complete Data Set' ;
run;

```

Fig. 2.18. SAS Example B6: Program

- a. A numeric variable **percent** that will contain the percent of population with driving licenses in each state
- b. A numeric variable **fuel** that measures the per capita motor fuel consumption in gallons in each state
- c. An ordinal variable called **incomgrp** assigned the value 1, 2, or 3 according to whether the per capita income (in thousands of dollars) is less than or equal to 3.8 , greater than 3.8 and less than or equal to 4.4, or over 4.4, respectively
- d. A nominal variable called **taxgrp** with a value of 'Low ' when the fuel tax is less than 8 cents and a value of 'High' otherwise
- e. A character variable named **state** containing the state name in uppercase and lowercase, for example, Kansas

A **format** statement ensures that values of the variable created in (a) are printed rounded to one decimal place and those of the variable created in (b) are printed as whole numbers (i.e., appropriate print formats are associated with **percent** and **fuel** variables). A **drop** statement is used to exclude variables **fuelc** and **st** from the data set created. Printed output from the program, a listing of the data set, is not reproduced here.

2.2.1 The UNIVARIATE procedure

Although there are several SAS procedures that produce descriptive statistics, **proc univariate** is best suited for studying the empirical distributions of variables in a data set. It produces a variety of descriptive statistics such as moments and percentiles and optionally creates output SAS data sets containing selected sample statistics. In addition, **proc univariate** can be used to produce high-resolution graphics such as histograms with overlaid kernel density estimates, quantile-quantile plots, and probability plots supplemented with goodness-of-fit statistics for a variety of distributions. A discussion of the statements that produce high-resolution graphics is deferred until Chapter 3. In this subsection, a brief discussion of several statements available for calculating sample statistics and saving those in a SAS data set are presented. This is followed by an illustrative example. The general structure of a **proc univariate** step (that includes five of the procedure information statements to be illustrated) is

```
PROC UNIVARIATE < options > ;
  BY variables ;
  CLASS variable-1 <(v-options)> < variable-2 <(v-options)> >
    ...< / KEYLEVEL= value1 | ( value1 value2 ) >;
  VAR variables ;
  ID variables ;
  OUTPUT < OUT=SAS-data-set >
    < keyword1=names...keywordk=names > < percentile-options >;
```

A large number of **proc** statement options are available for **proc univariate**. Although some of these are standard options such as the **data=** option for naming the data set to be analyzed or the **noprint** for suppressing printed output, others are more specialized. Some of these special **proc** statement options are summarized below.

Some PROC Statement Options

alpha= option specifies an α for calculating $(1 - \alpha)100\%$ confidence intervals, the default being .05

cibasic < (< type= ><alpha= >) > option calculates $(1 - \alpha)100\%$ confidence intervals for the mean, standard deviation, and variance assuming

that the data are normally distributed. Optionally, **type** may be set equal to one of the keywords **lower**, **upper**, or **two sided**. The defaults are **type=twosided** and **alpha** value set in the above **alpha=** proc option or the default value of .05.

mu0= option is used to list value(s) (μ_0) stipulated in the hypotheses for tests concerning population means corresponding to the variables listed in the **var** statement. The tests performed are the Student's *t*-test, the sign test, and the Wilcoxon signed rank test.

normal option requests tests for normality. Computed test statistics and *p*-values for the Shapiro-Wilk test (for sample sizes less than or equal to 2000), the Kolmogorov-Smirnov test, the Anderson-Darling test, and the Cramér-von Mises test are output.

pctldef= gives the user the option of selecting one of five methods (labeled 1, 2, 3, 4, or 5) that **proc univariate** uses for calculating sample percentiles. These methods depend on the sample size *n* and the percentile *p* and are described in the documentation. The default method is 5.

plots option requests that low-resolution descriptive graphics (a histogram or a stem-and-leaf plot, a box plot, and a normal probability plot) are to be produced.

trim=values < (<type= ><alpha= >) > option requests the computation of trimmed means for each variable in the **var** list, where the *values* list is the numbers *k* or the fractions *p* of observations to be *trimmed* from both ends of the observations ordered smallest to largest. If *p* are specified, then the numbers trimmed equal *np* rounded up to the nearest integer, respectively. Confidence intervals for the population means are also calculated based on the trimmed means and estimates of their standard errors; the options **type=** and **alpha=** may be used to change their default settings, as described for the **cibasic** proc option earlier.

vardef= specifies the divisor to be used in the calculation of variance and standard deviation. The default value for the divisor is **df** when the degrees of freedom $n - 1$ will be used. Other possible values that may be specified are **n**, **wdf**, and **weight** or **wgt**, respectively, when the sample size *n*, the weighted degrees of freedom $\sum w_i - 1$, or the sum of weights $\sum w_i$ (where w_i are the weights specified in **weight** statement) will be used.

Some CLASS Statement Options

The variables list in the **class** statement specifies groups into which the observations in a data set are classified into for the purpose of calculating statistics.

The values of these variables can be numeric or character and are called *levels*. For the purpose of displaying output from such an analysis (e.g., tables), procedures such as **univariate** must be provided with a way to determine in what order the statistics calculated for each level of a class variable are to be displayed. In many procedures, the **order=** option is available as a **proc** statement option to be used for this purpose. In **proc univariate**, this option is available as one of the *v-options* in the **class** statement, so each level of each class variable may be separately ordered.

The **class** statement allows the *v-options* **missing** and **order=** to be specified, enclosed in parentheses, for each of the variables in the *class variable list*. For example, using the **order=** option for each variable allows the user to specify the display order of the levels of each of the class variables separately. The default setting for the **order=** option is **internal**, which specifies that the internal unformatted (character or numeric) value of a variable be used for this purpose. The other available choices are **data**, in which case the levels will be displayed in the order they appeared in the input data, **formatted**, which requests that the levels be ordered by their formatted values, and **freq**, which requests that levels be listed in the decreasing order of frequency of observations for each level.

```
libname mylib 'C:\Documents and Settings\...\stat479\';

proc univariate data=mylib.fueldata plots normal;
  var pop income;
  id state;
  title 'Use of Proc Univariate to Examine Distributions:1';
run;

proc univariate data=mylib.fueldata cibasic mu0=4 500 trim=2;
  var income fuel;
  id state;
  title 'Use of Proc Univariate to Compute Statistics:2';
run;

proc univariate data=mylib.fueldata noprint;
  var fuel percent;
  output out=stats pctlpts=33.3 66.7 pctlpre=fuel lic;
  title 'Calculation of User Specified Percentile Points';
run;

proc print data=stats;
run;
```

Fig. 2.19. SAS Example B7: Program

SAS Example B7

Several variables in the SAS data set created in SAS Example B6 on fuel consumption data are analyzed using **proc univariate** in the SAS Example B7 program (see Fig. 2.19) to illustrate the use of the procedure options and statements discussed in this section. The previously saved SAS data set named

Use of Proc Univariate to Examine Distributions:1				1
The UNIVARIATE Procedure				
Variable: pop (Population(in thousands))				
Moments				
N	48	Sum Weights	48	
Mean	4296.91667	Sum Observations	206252	
Std Deviation	4441.1087	Variance	19723446.5	
Skewness	2.00893087	Kurtosis	4.32297497	
Uncorrected SS	1813249642	Corrected SS	927001986	
Coeff Variation	103.355709	Std Error Mean	641.018826	
Basic Statistical Measures				
Location		Variability		
Mean	4296.917	Std Deviation	4441	
Median	2982.500	Variance	19723447	
Mode	.	Range	20123	
		Interquartile Range	3894	
Tests for Normality				
Test	--Statistic---		-----p Value-----	
Shapiro-Wilk	W	0.771583	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.208119	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.591415	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	3.36823	Pr > A-Sq	<0.0050

Fig. 2.20. SAS Example B7: Summary statistics and tests for Normality

fueldat is accessed using the two-level name mylib.fueldat. In the first proc step in this program, the options plots and normal produce line-printer style

Use of Proc Univariate to Examine Distributions:1			2
The UNIVARIATE Procedure			
Variable: pop (Population(in thousands))			
Quantiles (Definition 5)			
Quantile	Estimate		
100% Max	20468.0		
99%	20468.0		
95%	11926.0		
90%	11251.0		
75% Q3	4989.0		
50% Median	2982.5		
25% Q1	1095.5		
10%	579.0		
5%	527.0		
1%	345.0		
0% Min	345.0		

Fig. 2.21. SAS Example B7: Quantiles

Extreme Observations					
-----Lowest-----			-----Highest-----		
Value	state	Obs	Value	state	Obs
345	Wyoming	40	11251	Illinois	12
462	Vermont	3	11649	Texas	37
527	Nevada	45	11926	Pennsylvania	9
565	Delaware	22	18366	New York	7
579	South Dakota	19	20468	California	48

Fig. 2.22. SAS Example B7: Extreme values

low-resolution stem-and-leaf plots, box plots, and normal probability plots of the population and income variables, named (**pop** and **income**), descriptive statistics including percentiles and extreme values, and several test statistics for testing normality. Those output for the population variable are reproduced in Figs. 2.20-2.24.

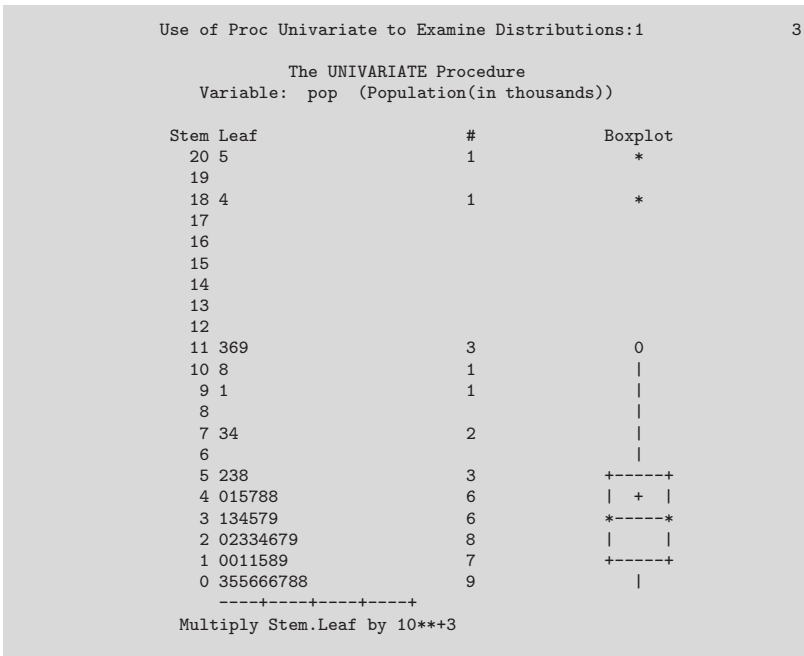


Fig. 2.23. SAS Example B7: Stem-and-leaf and box plots

Note that the percentiles shown in Fig. 2.21 are calculated using **definition 5**. The lowest and highest five extreme values, also output in page 2,

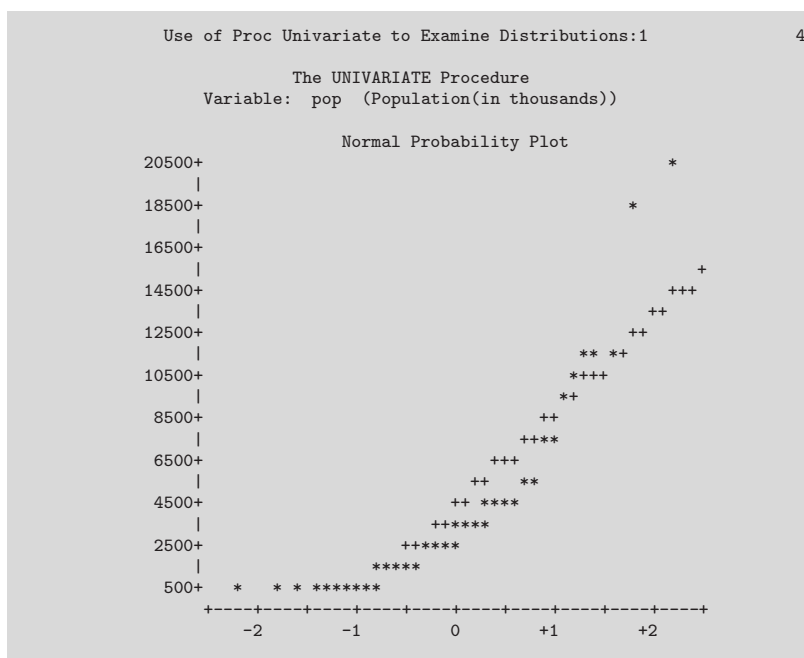


Fig. 2.24. SAS Example B7: Normal probability plot

are shown in Fig. 2.22. The `id state;` statement results in these values being identified by the corresponding state name. The stem-and-leaf plot and the box plot (see Fig. 2.23) show a highly positively skewed population distribution with two extreme values, which were identified as those that correspond to the states of New York and California in Fig. 2.22.

The points (plotted as asterisks) in the normal probability plot (see Fig. 2.24) show a clear bowl-shaped pattern, indicating a right-skewed distribution for the data. The interpretation of normal probability plots is discussed in more detail in Section 3.5 of Appendix A. Using the Shapiro-Wilk statistic, the null hypothesis of normality is rejected, as the p -value is $< .0001$ (see Fig. 2.20), thus substantiating the evidence from the graphical displays. Note, too, that the `skewness` statistic (which is expected to be near zero for symmetrical distributions) is quite large here.

The second proc step analyzes the distribution of the variables `income` and `fuel` and includes the procedure options `cibasic`, `mu0=4 500`, and `trim=2`. By default, the `cibasic` option produces 95% confidence intervals for the population mean μ , the population standard deviation σ , and the population variance σ^2 for each of the variables calculated under the normality assumption for the data. These are shown in Fig. 2.25 for the `income` variable (recall that the data values are per capita income figures in thousands of dollars).

Basic Statistical Measures				9
Location		Variability		
Mean	4.241833	Std Deviation	0.57362	
Median	4.298000	Variance	0.32904	
Mode	5.126000	Range	2.27900	
		Interquartile Range	0.85050	
Basic Confidence Limits Assuming Normality				
Parameter	Estimate	95% Confidence Limits		
Mean	4.24183	4.07527	4.40840	
Std Deviation	0.57362	0.47752	0.71851	
Variance	0.32904	0.22803	0.51626	
Tests for Location: Mu0=4				
Test	-Statistic-	-----p Value-----		
Student's t	t 2.920853	Pr > t	0.0053	
Sign	M 7	Pr >= M	0.0595	
Signed Rank	S 248.5	Pr >= S	0.0093	

Fig. 2.25. SAS Example B7: Confidence intervals and tests for the `income` variable

The confidence interval reported for μ is (4.07527, 4.40840) and the p -value for the two-sided t -test of $H_0 : \mu = 4$ vs. $H_a : \mu \neq 4$ is calculated to be .0053. The p -value for the corresponding one-sided test is, of course, .0053/2 = .0026. The tests and confidence intervals are not reproduced here for the `fuel` variable, but they are to be similarly interpreted.

Use of Proc Univariate to Compute Statistics:2						10
The UNIVARIATE Procedure						
Variable: income (Per Capita Income(in thsnds.))						
Trimmed Means						
Percent Trimmed in Tail	Number Trimmed in Tail	Trimmed Mean	Std Error Trimmed Mean	95% Confidence Limits		DF
4.17	2	4.239795	0.087055	4.064233	4.415358	43
Trimmed Means						
Percent Trimmed in Tail	t for H0: Mu0=4.00		Pr > t			
4.17	2.754533		0.0086			

Fig. 2.26. SAS Example B7: Trimmed means for the `income` variable

Calculation of User Specified Percentile Points					15
Obs	fuel33_3	fuel66_7	lic33_3	lic66_7	
1	524.994	609.991	54.4421	58.0087	

Fig. 2.27. SAS Example B7: Calculating user-specified percentiles

The estimate, confidence interval, and associated *t*-test for the mean μ under trimming for the `income` variable appear in Fig. 2.26. The option `trim=2` requested that the `trimmed mean` be computed after the *2 smallest* and the *2 largest* observations are deleted from the sample, which is equivalent to approximately 4% trimming from the tails of the distribution of the `income` variable. Associated confidence intervals and a *t*-test for the population mean μ is computed based on the standard error of the trimmed mean. For a symmetric distribution, the trimmed mean is an unbiased estimate of the population mean. The results under trimming here indicate that the estimates and test statistics are not significantly different from those statistics calculated from the complete sample (see Figs. 2.25 and 2.26). This is also an indicator of the symmetry of the population distribution of the `income` variable.

The final `proc` step requests the calculation of the 33.3 and 66.7 percentiles of the `fuel` (Fuel Consumption (in gallons/person)) and `percent` (% of Population with Driving Licenses) variables. In this example, the printed output is suppressed (as a result of the `noprint` `proc` option) and the output is written to a new SAS data set named `stats`. The percentiles that cannot be directly requested via the usage of a keyword such as `p1`, `p10`, or `p90` are calculated by the use of the pair of keywords `pctlpts=` and `pctlpre=` used concurrently. For example, the use of `pctlpts=33.3 66.7 pctlpre=fuel lic` generates the 33.3 and 66.7 percentiles for the two analysis variables (variables in the `var` statement) and adds them to the new data set as values of new variables named `fuel33_3`, `fuel66_7`, `lic33_3`, and `lic66_7`, respectively. The `proc print data=stats;` statement produces the output of these values shown in Fig. 2.27.

2.2.2 The FREQ procedure

The `freq` procedure in SAS computes many statistics and measures related to the analysis of categorical data. The discussion in this subsection is primarily intended to illustrate the use of statements and options to generate these statistics rather than a presentation of statistical methodology involved in the analysis of categorical data. It is recommended that the prospective user of `proc freq` consult references cited to learn more about techniques available for hypothesis testing and measuring association among categorical variables. Moreover, the type of inference depends on many factors such as sampling

strategy; thus, a knowledge of how the data are collected is also necessary for making relevant conclusions.

A chi-square goodness-of-fit test can be used to test several types of hypothesis using frequency counts. For example, using a one-way frequency table with k classes, one could compute a chi-square statistic to test whether the counts conform to sampling from a multinomial population with specified probabilities. In this case, the null hypothesis of interest is

$$H_0 : p_i = p_{i0}, \quad i = 1, 2, \dots, k$$

where the p_{i0} 's are postulated values of the multinomial probabilities. The Pearson chi-square statistic is given by

$$\chi^2 = \sum_{i=1}^k \frac{(f_i - e_i)^2}{e_i}$$

where f_i is the observed frequency count in class i and e_i is the expected frequency calculated under the null hypothesis (i.e., $e_i = p_{i0}N$ where N is the total number of responses). Another application of a chi-square test is for testing *homogeneity* of several multinomial populations. In this case, random samples are taken from each population and then classified by a categorical variable. The populations are usually defined by levels of variables such as gender, age group, state, etc., and the levels of the categorical variable form the k categories of the multinomial populations.

For example, suppose samples are drawn from two populations (say, males and females or persons below and above the age of 40) and they are grouped into three categories (say, according to three levels of support for a certain local bond issue). Suppose that the multinomial probabilities for each population are as given in the following table:

	Groups		
Populations	p_{11}	p_{12}	p_{13}
	p_{21}	p_{22}	p_{23}

Then the null hypothesis of homogeneity of populations (i.e., whether random samples were drawn from the same multinomial population) is given by

$$H_0 : p_{11} = p_{21}, \quad p_{12} = p_{22}, \quad p_{13} = p_{23}$$

Note carefully that the sampling procedure here is different from the process used in the construction of a *contingency table*. In the above situation, random samples are drawn from two different populations and then each sample is classified into three different groups. Contingency tables are constructed by multiple classification of a single random sample. Observations in a sample may be cross-classified by variables with ordinal or nominal data values defining *categorical variables*. These variables may be covariates already present in

the data set (e.g., gender, marital status, or region), thus forming natural subsets or strata of the data or may be generated from other quantitative variables such as **population** or **income**. For example, observations in a sample may be categorized into three income groups (say “low,” “middle,” or “high”) by creating a new variable, say **incgrp**, and assigning the above character strings as its values according to whether the value of the income variable is below \$30,000, between \$30,000 and \$70,000, or above \$70,000, respectively.

A chi-square statistic can be computed for a two-way $r \times c$ contingency table to test whether the two categorical variables are independent; that is, the null hypothesis tested is of the form

$$H_0 : p_{ij} = p_{i.}p_{.j}, \quad i = 1, 2, \dots, r, \quad j = 1, 2, \dots, c$$

where the p_{ij} , are the probabilities considering that the entire sample is from a multinomial population, and $p_{i.}$ and $p_{.j}$, called the *marginal* probabilities, are probabilities for multinomial populations defined by each categorical variable. The chi-square statistic for the test of this hypothesis is given by

$$\chi^2 = \sum_i \sum_j \frac{(f_{ij} - e_{ij})^2}{e_{ij}}$$

where f_{ij} is the observed frequency in the ij th cell and $e_{ij} = f_{i.}f_{.j}/N$, where $f_{i.}$ and $f_{.j}$ are the observed row and column marginal frequencies, respectively.

When the row and column variables are independent, the above statistic has an asymptotic chi-square distribution with $(r-1)(c-1)$ degrees of freedom. Instead of χ^2 , the likelihood ratio chi-square statistic, usually denoted by G^2 , that has the same asymptotic null distribution may be computed. If the row and columns are ordinal variables, the Mantel-Haenszel chi-square statistic tests the alternative hypothesis that there is a linear association between them. Fisher’s exact test is another test of association between the row and column variables that does not depend on asymptotic theory. It is thus suitable for small sample sizes and for sparse tables. One may compute measures of association between variables that may or may not depend on the chi-square test of independence. Some of these are illustrated in SAS Example B8.

One would use **proc freq** to analyze count data using one-way frequency tables or two-way or higher-order contingency tables. In addition to chi-square statistics for testing whether two categorical variables are independent, for a two-way contingency table, **proc freq** also computes measures of association that estimate the strength of association between the pair of variables. In this subsection, a brief discussion of several statements available in **proc freq** followed by an illustrative example are presented. The general structure of a **proc freq** step is

```
PROC FREQ < options > ;
  BY variables ;
  TABLES requests < / options > ;
```

```

EXACT statistic-options < / computation-options > ;
TEST options ;
OUTPUT < OUT=SAS-data-set > options ;

```

The primary statement in `proc freq` is the `tables` statement for requesting tables having different structures with options for selecting statistics to be included in those tables. Since the computation of frequencies requires that the variables used in the `tables` statement are necessarily discrete-valued (containing either nominal or ordinal data) such as category, classification, or grouping type variables, statements such as `var` or `class` are not available in `proc freq`.

The syntax of the `tables` statement allow the user to request one-way tables just by listing the variables in the `tables` statement, and two-way tables by two variable names combined with an asterisk between them. Thus, the statement `tables region;` produces one-way tables with frequency counts of observations for each *level* of `region` and the statement `tables taxgrp*region;` produces a two-way cross-tabulation with the *levels* of the variable `taxgrp` as the rows of the table and the *levels* of `region` as the columns. A combination of a level of `taxgrp` and a level of `region` forms a *cell* in the table. In this case, frequencies of observations are tallied for every possible combination of the two variables and entered in the respective cells; they are called *cell frequencies*. Multiway combinations of variables such as `p*q*r*s` are possible in which case two-way tables are produced for every combination of levels of each of the variables `p` and `q`. The cells in each two-way table are formed by combinations of a level of variables `r` and `s`. The `proc` statement option `page` may be used to force these tables to be output on different pages. The `tables` statement syntax also allows variations such as `tables q*(r s)`, which is equivalent to the specification `tables q*r q*s`, or `tables (p q)*(r s)`, which is equivalent to `tables p*r q*r p*s q*s`.

By default, one-way frequency tables contain the statistics frequency, cumulative frequency, percentage frequency, and cumulative percentage computed for each level of the variable and a two-way or multi-way tables may include cell frequency, cell percentage of the total frequency, cell percentage of row frequency, and cell percentage of column frequency computed for each cell. Many options are available with the `tables` statement to control the statistics that are calculated and output by `proc freq`. While some of these are simple options for suppressing the statistics computed by default, others request additional statistics such as goodness-of-fit statistics and measures of association to be computed. An abbreviated description of these options is provided below.

Some TABLES Statement Options

`nocol` suppresses printing the column percentage for each cell.

`nocum` suppresses printing the cumulative frequencies and cumulative percentages in one-way frequency tables and in list format.

`norow` suppresses printing the row percentage for each cell.

`nopercent` suppresses printing the percentage, row percentage, and column percentage in two-way tables, or percentages and cumulative percentages in one-way tables and in list format.

`noprint` suppresses printing the frequency table but displays other statistics.

`list` prints multiway tables in list format.

`binomial` requests binomial proportion, confidence limits and test for one-way tables.

`testf=` specifies expected frequencies for a one-way table chi-square test.

`testp=` specifies expected proportions for a one-way table chi-square test.

`chisq` requests chi-square tests and measures of association based on chi-square.

`cellchi2` prints each cell's contribution to the total Pearson chi-square statistic.

`deviation` prints the deviation of the cell frequency from the expected value for each cell.

`expected` prints the expected cell frequency for each cell under the null.

`fisher` requests Fisher's exact test for tables larger than 2×2 .

`measures` requests measures of association and their asymptotic standard errors.

`cl` requests confidence limits for the measures statistics.

`alpha=` sets the confidence level for confidence limits.

`agree` requests tests and measures of classification agreement.

Options such as `binomial`, `testf=`, and `testp=` are used for specifying either the postulated probabilities (p_{i0} 's) where $H_0 : p_i = p_{i0}$, $i = 1, 2, \dots, k$, or the expected frequencies in a sample of size n classified in a one-way frequency table for performing a chi-square goodness-of-fit test. An application is provided as an exercise at the end of this chapter.

SAS Example B8

In the following contrived example of a two-way table, suppose that subjects are classified according to levels of two variables A and B. The column variable A has three categories, say a_1, a_2 and a_3 and the row variable B has three categories, say b_1, b_2 , and b_3 . Most often, the row variable is called the *dependent variable* if the categories of the variable are recognized as possible *outcomes* or *responses*. An example would be where factor B is **Marital Status** and factor A is a response to a question with three possibilities. Consider the table of frequencies:

	b_1	b_2	b_3	Total
a_1	8	16	31	55
a_2	9	18	74	101
a_3	34	23	17	74
Total	51	57	122	230

In this case, the column variable is called the independent variable with categories being *classes*, *groups*, or *strata*. The designation of whether the two types of variable are assigned to rows or columns is usually a matter of choice.

In the above setup, subjects from each of the column categories (independent variable) can be viewed as being classified into one of the row categories (dependent variable). The choice of the independent and dependent variables does not affect the statistical analysis of the data except when part of the inference is measuring the predictability of a response category given that an object belongs to a certain group or class. Otherwise, when variables cannot be clearly identified as independent and dependent variables, statistics and measures unaffected by an arbitrary designation are preferred.

In the SAS Example B8 program (see Fig. 2.28), the cell frequencies are directly input to `proc freq` instead of raw data (which are not available in this example). The use of the `weight` statement allows SAS to construct the two-way cross-tabulation using the cell counts. In the first part of the output (see Fig. 2.29), only the statistics observed count f_i , the expected frequency e_i , and its contribution to the total chi-squared statistic are displayed in each cell (i.e., percentage of the total frequency, percentage of row frequency, and percentage of column frequency are suppressed using `tables` statement options given earlier). It is clear that the cells (2,1), (2,3), (3,1), and (3,3) provide the largest contributions to the total chi-squared statistic of 52.4. It is observed that at the lowest level of B, the response is smaller than expected for group 2 of A and larger than expected for group 3 of A. This pattern is reversed at the highest level of B.

```

options formchar="|----|+|----+|-/\<>*";
data ex8;
input A $ B $ count @@;
datalines;
a1 b1 8 a1 b2 16 a1 b3 31
a2 b1 9 a2 b2 18 a2 b3 74
a3 b1 34 a3 b2 23 a3 b3 17
;
run;
proc freq;
weight count;
tables A*B/chisq expected cellchi2 nocol nopercnt norow measures;
title 'Example B8: Illustration of Tables Options';
run;

```

Fig. 2.28. SAS Example B8: Program

For two-way frequency tables, the chi-square test of independence is a test of *general association*, where the null hypothesis is that the row and column variables are independent (no association) and the alternative hypothesis is that an association exists between the two variables with the type of association unspecified. The chi-square statistic and the likelihood ratio statistic are both suitable for testing this hypothesis. `proc freq` computes these statistics in response to `chisq` option in the `tables` statement. For large sample sizes and if the null hypothesis is true, these test statistics have approximately a chi-square distribution. (For small samples, the user may request that Fisher's exact test be computed by specifying the `exact` option in the `tables` statement.) In Fig. 2.29, the p -values of both the chi-square statistic and the likelihood ratio statistic are smaller than, say .01. Thus, the null hypothesis that the two variables are independent is rejected, leading to the conclusion that there is some type of association between these two variables.

The three measures `phi coefficient` ϕ , `contingency coefficient` C , and `Cramer's V` displayed next in the output are suitable for measuring the strength of the dependency between nominal variables but are also applicable for ordinal variables. The value of C is zero if there is no association between the two variables but has a value that is less than 1 even with perfect dependence. Its value is dependent on the size of the table with a maximum value of $\sqrt{(r-1)/r}$ for an $r \times r$ table. For a 3×3 table, this value is 0.816. Thus, the value of 0.43 of C appears to indicate a strength midway between no association and a perfect association.

The other statistics also lead to similar conclusions. Cramer's V is a normed measure, so its value is between 0 and 1; thus, a value of 0.34 is approximately in the bottom third of the scale. The range for ϕ is $0 < \phi < \min\{\sqrt{r-1}, \sqrt{c-1}\}$. Since for this table the maximum is 1.4, a strength of association similar to the above measures is indicated by a value of 0.48 for ϕ .

Example B8: Illustration of Tables Options					1
The FREQ Procedure					
Table of A by B					
A	B				
Frequency					
Expected					
Cell Chi-Square	b1	b2	b3	Total	
a1	8	16	31	55	
	12.196	13.63	29.174		
	1.4434	0.4119	0.1143		
a2	9	18	74	101	
	22.396	25.03	53.574		
	8.0124	1.9747	7.7878		
a3	34	23	17	74	
	16.409	18.339	39.252		
	18.859	1.1846	12.615		
Total	51	57	122	230	
Statistics for Table of A by B					
Statistic	DF	Value	Prob		
Chi-Square	4	52.4031	<.0001		
Likelihood Ratio Chi-Square	4	53.1793	<.0001		
Mantel-Haenszel Chi-Square	1	25.0216	<.0001		
Phi Coefficient		0.4773			
Contingency Coefficient		0.4308			
Cramer's V		0.3375			

Fig. 2.29. SAS Example B8: $A \times B$ chi-square test

The above three measures of association are all derived from the Pearson chi-square statistic. There are many other measures of association between two categorical variables available and `proc freq` calculates several of these. Some of these statistics are briefly discussed here. Many of these statistical measures also require the assignment of a dependent variable and an independent variable, as the goal is to predict a rank (category) of an individual on the dependent variable given that the individual belongs to a certain category in the independent variable.

For calculating the following measures for the two variables under consideration, pairs of observations are first classified as concordant or discordant. A pair is concordant if the observation with the larger value for variable one also has the larger value for variable two, and it is discordant if the observation with the larger value for variable one has the smaller value for variable two. Thus, the pair of observations (12, 2.7) and (15, 3.1) are concordant and the pair (12, 2.7) and (10, 3.1) are discordant.

Example B8: Illustration of Tables Options			2
The FREQ Procedure			
Statistics for Table of A by B			
Statistic	Value	ASE	
Gamma	-0.4375	0.0828	
Kendall's Tau-b	-0.2981	0.0586	
Stuart's Tau-c	-0.2804	0.0555	
Somers' D C R	-0.2891	0.0575	
Somers' D R C	-0.3074	0.0601	
Pearson Correlation	-0.3306	0.0626	
Spearman Correlation	-0.3354	0.0650	
Lambda Asymmetric C R	0.1574	0.0607	
Lambda Asymmetric R C	0.2326	0.0622	
Lambda Symmetric	0.1983	0.0540	
Uncertainty Coefficient C R	0.1138	0.0293	
Uncertainty Coefficient R C	0.1082	0.0281	
Uncertainty Coefficient Symmetric	0.1109	0.0286	
Sample Size = 230			

Fig. 2.30. SAS Example B8: $A \times B$ measures of association

Gamma is a normed measure of association based on the numbers of concordant and discordant pairs. If there are no discordant pairs, **Gamma** is +1 and perfect positive association exists between the two variables and if there are no concordant pairs, **Gamma** is -1 and perfect negative association exists between the two variables. Values in between -1 and +1 measure the strength of negative or positive association. If the numbers of discordant and concordant pairs are equal, **Gamma** is zero and the rank of the independent variable cannot be used to predict the rank of the dependent variable. In the SAS Example B8 output (see Fig. 2.30), **Gamma** = -0.4375 with an estimated asymptotic standard error (ASE) of 0.0828, indicating a negative association.

Kendall's tau-b is the ratio of the difference between the number of concordant and discordant pairs to the total number of pairs. It is scaled to be between -1 and +1 when there are no ties, but not otherwise. The ordinal measure **Somers' D** on the other hand adjusts for ties by counting pairs where ties occur only on the independent variable so that the value of the statistic lies between -1 and +1 when such ties occur. Usually, two values of this statistic are computed: one when the row variable is considered the independent variable (**Somers' D C|R**) and one when the column is considered the independent variable (**Somers' D R|C**). The values differ because of the way ties are counted. In SAS Example B8, **Somers' D R|C** = 0.2326, showing a moderate positive association.

The nominal measure **asymmetric lambda**, $\lambda(R|C)$, is interpreted as the proportional improvement in predicting the dependent (row) variable given the independent (column) variable. Asymmetric lambda has the range $0 \leq \lambda(R|C) \leq 1$, although values around 0.3 are considered high. The measure $\lambda(C|R)$ may be interpreted similarly. In SAS Example B8, if information about variable B is used to predict A, the *proportional reduction in error* in the prediction is 23.26% compared to not using that information. **Stuart's tau c** makes an adjustment for table size in addition to a correction for ties. Tau-c is appropriate only when both variables lie on an ordinal scale. Tau-c also is in the range $-1 \leq \tau_c \leq 1$.

The **Pearson correlation coefficient** and the **Spearman rank-order correlation coefficient** are also appropriate for ordinal variables. The Pearson correlation describes the strength of the linear association between the row and column variables. It is computed using the row and column scores specified by the **scores=** option in the **tables** statement. By default, the row or column scores are the integers 1, 2,... for character variables and the actual variable values for numeric variables. Consult SAS documentation for other options. The Spearman correlation is computed with rank scores.

SAS Example B9

The analysis of the SAS data set on fuel consumption created in SAS Example B6 is continued in SAS Example B9 (see Fig. 2.31 for the program) using **proc freq** to illustrate the statistics resulting from some of the **tables** statement options discussed in this section. A new SAS data set is created by supplementing the original data set with two category variables, **fuelgrp** and **licgrp**, each with three levels, in the data step. The 33.3 and 66.7 percentiles of the **fuel** and **percent** variables calculated in SAS Example B7 (see Fig. 2.27) aid in the determination of cutoff values for creating the corresponding category variables. Thus, the category variables **fuelgrp** and **licgrp** will have three levels each. In addition, **proc format** described in Section 2.1.4 facilitates the creation of output formats to convert the ordinal levels of the three categorical variables **fuelgrp**, **incomgrp**, and **licgrp** to more descriptive strings when they are printed.

The **proc** step generates three contingency tables for combinations of the variable **fuelgrp** with each of **taxgrp**, **incomgrp**, and **licgrp**. The output is shown in Figs. 2.32, 2.33, and 2.34. The **fuelgrp** by **taxgrp** table is a 3×2 cross-tabulation where the levels of **fuelgrp** are ordered by their internal (unformatted) values of 1, 2, and 3. However, internal values of the two levels of **taxgrp** are the strings 'Low' and 'High'; thus, they are ordered by their alphanumeric values. The *p*-values of both the chi-square statistic and the likelihood ratio statistic are smaller than, say, .01. Thus, the null hypothesis that the two variables are independent is rejected.

```

options formchar="|----|+|----+|-/\\<>*";
libname mylib 'C:\Documents and Settings\mervyn\My Documents\Classwork\stat479\';

data fueldat2;
set mylib.fueldat;

if fuel=<525 then fuelgrp=1;
else if 525<fuel=<610 then fuelgrp=2;
else fuelgrp=3;

if percent=<54 then licgrp=1;
else if 54<percent=<58 then licgrp=2;
else licgrp=3;

label licgrp='% Driving Licenses'
      fuelgrp='Fuel Consumption';
run;

proc format;
  value lg 1='below 54%'
          2='54 to 58%'
          3='above 58%' ;
  value fg 1 = 'Low Fuel Use'
          2 = 'Medium Fuel Use'
          3 = 'High Fuel Use';
  value ing 1 = 'Low Income'
            2 = 'Middle Income'
            3 = 'High Income';
run;

proc freq data=fueldat2;
  tables fuelgrp*(taxgrp incomgrp)/chisq expected
          cellchi2 nocol nopercent norow;
  tables fuelgrp*licgrp/chisq expected nocol nopercent norow measures;
  tables taxgrp*fuelgrp/list ;
  format fuelgrp fg. licgrp pg. incomgrp ing. ;
  title 'Output from Proc Freq';
run;

```

Fig. 2.31. SAS Example B9: Program

To study the strength of association between the two variables **fuelgrp** and **taxgrp**, the statistics in Fig. 2.32 are used here. The three measures **phi coefficient** ϕ , **contingency coefficient** C , and **Cramer's V** displayed next in the output are suitable statistics for measuring the strength of the dependency between nominal variables and are also applicable for ordinal variables, as in this example. The value of C is zero if there is no association between the two variables but has a value that is less than 1 even with perfect dependence. Its value is dependent on the size of the table with a maximum value of $\sqrt{(r-1)/r}$ for an $r \times r$ table. For a 3×3 table, this value is 0.816. Thus, the value of 0.40 for C appears to indicate a strength of about 50% of a perfect association.

The other statistics in Fig. 2.32 also lead to similar conclusions. Cramer's V is a normed measure, so its value is between 0 and 1; thus, a value of 0.44 is about in the middle of the scale. The range for ϕ is $0 < \phi <$

$\min\{\sqrt{r-1}, \sqrt{c-1}\}$. Thus, for this table, the maximum is 1, so again a strength of association similar to the above measures is indicated. The cross-tabulation **fuelgrp** by **incomgrp** shown in Fig. 2.33 is a 3×3 table. Again, the chi-square and the likelihood ratio statistic are both significant at the .01 level, indicating dependency. A study of the values for the three measures discussed above, shown in Fig. 2.33, indicates a slightly stronger association between the variables **fuelgrp** and **incomgrp**.

The hypothesis of independence between the two variables **fuelgrp** and **licgrp** is rejected at .05 (the likelihood ratio statistic has a p -value of .0026; see Fig. 2.34). This is one situation where Fisher's exact test may be performed instead of the chi-square test because conditions for that test are clearly not met due to several small cell frequencies. In this example, the inclusion of the **exact** option produces the output in Fig. 2.35. The p -value is smaller than .05 so the conclusion is that the independence hypothesis is rejected at $\alpha = .05$

Output from Proc Freq				1
The FREQ Procedure				
Table of fuelgrp by taxgrp				
fuelgrp(Fuel Consumption)				
	taxgrp(Fuel Tax)			
Frequency				
Expected				
Cell Chi-Square	High	Low	Total	
-----+-----+-----+				
Low Fuel Use	10	6	16	
	7.3333	8.6667		
	0.9697	0.8205		
-----+-----+-----+				
Medium Fuel Use	10	7	17	
	7.7917	9.2083		
	0.6259	0.5296		
-----+-----+-----+				
High Fuel Use	2	13	15	
	6.875	8.125		
	3.4568	2.925		
-----+-----+-----+				
Total	22	26	48	
Statistics for Table of fuelgrp by taxgrp				
Statistic	DF	Value	Prob	
-----+-----+-----+				
Chi-Square	2	9.3275	0.0094	
Likelihood Ratio Chi-Square	2	10.2233	0.0060	
Mantel-Haenszel Chi-Square	1	7.2412	0.0071	
Phi Coefficient		0.4408		
Contingency Coefficient		0.4034		
Cramer's V		0.4408		
Sample Size = 48				

Fig. 2.32. SAS Example B9: Fuel group \times Tax group cross-tabulation

Output from Proc Freq

2

The FREQ Procedure

Table of fuelgrp by incomgrp

fuelgrp(Fuel Consumption)		incomgrp(Per capita Income)			
Frequency					
Expected					
Cell Chi-Square	Low Inco	Middle I	High Inc	Total	
	me	ncome	ome		
Low Fuel Use	1	3	12	16	
	4.3333	6	5.6667		
	2.5641	1.5	7.0784		
Medium Fuel Use	8	7	2	17	
	4.6042	6.375	6.0208		
	2.5046	0.0613	2.6852		
High Fuel Use	4	8	3	15	
	4.0625	5.625	5.3125		
	0.001	1.0028	1.0066		
Total	13	18	17	48	

Statistics for Table of fuelgrp by incomgrp

Statistic	DF	Value	Prob
Chi-Square	4	18.4040	0.0010
Likelihood Ratio Chi-Square	4	18.7393	0.0009
Mantel-Haenszel Chi-Square	1	7.2622	0.0070
Phi Coefficient		0.6192	
Contingency Coefficient		0.5265	
Cramer's V		0.4378	

WARNING: 33% of the cells have expected counts less than 5. Chi-Square may not be a valid test.

Sample Size = 48

Fig. 2.33. SAS Example B9: Fuel group \times Income group cross-tabulation

If it is concluded that there is association between the two variables, several statistics are available for evaluating the strength of such association. The output resulting from including the `measures` option in a `tables` statement is shown in Fig. 2.36. For the interpretation of Γ and τ_b , consider `licgrp` as the independent variable that is used to predict the dependent variable `fuelgrp` and that both variables are ordinal. The value of 0.5641 for Γ indicates a positive association between the two variables. This implies that the ordering of the ranks of states for these two variables are positively correlated and that if the orders of ranks of percentage of licenses is used to predict the orders of ranks of fuel use, the proportional reduction in error compared to randomly assigning ranks of fuel use for pairs of states is 56%. The ordi-

Output from Proc Freq					3
The FREQ Procedure					
Table of fuelgrp by licgrp					
fuelgrp(Fuel Consumption)		licgrp(% Driving Licenses)			
Frequency					
Expected					
		below 54	54 to 58	above 58	Total
		%	%	%	
Low Fuel Use	6	9	1		16
	4.6667	6	5.3333		
Medium Fuel Use	7	5	5		17
	4.9583	6.375	5.6667		
High Fuel Use	1	4	10		15
	4.375	5.625	5		
Total	14	18	16		48
Statistics for Table of fuelgrp by licgrp					
Statistic	DF	Value	Prob		
Chi-Square	4	14.6905	0.0054		
Likelihood Ratio Chi-Square	4	16.2966	0.0026		
Mantel-Haenszel Chi-Square	1	9.9989	0.0016		
Phi Coefficient		0.5532			
Contingency Coefficient		0.4841			
Cramer's V		0.3912			
WARNING: 33% of the cells have expected counts less than 5. Chi-Square may not be a valid test.					

Fig. 2.34. SAS Example B9: Fuel group \times Licenses group cross-tabulation

nal measures of association Kendall's τ_b and Somers' $D|C$ both have a value of 0.40 again confirming the positive association between these two variables. Rather than being concerned with the ranking of pairs of observations on the two variables (concordancy or discordancy) as discussed previously, Spearman's ρ is measures the strength of the relationship between the overall ranks of each observation (or subject) on the two variables.

Fisher's Exact Test	
Table Probability (P)	5.544E-06
Pr <= P	0.0046
Sample Size = 48	

Fig. 2.35. SAS Example B9: Fisher's exact test for Fuel \times Licenses

Output from Proc Freq			4
The FREQ Procedure			
Statistics for Table of fuelgrp by licgrp			
Statistic	Value	ASE	
Gamma	0.5641	0.1272	
Kendall's Tau-b	0.4024	0.0983	
Stuart's Tau-c	0.4010	0.0988	
Somers' D C R	0.4016	0.0979	
Somers' D R C	0.4031	0.0989	
Pearson Correlation	0.4612	0.1049	
Spearman Correlation	0.4640	0.1090	
Lambda Asymmetric C R	0.2667	0.1456	
Lambda Asymmetric R C	0.2903	0.1463	
Lambda Symmetric	0.2787	0.1335	
Uncertainty Coefficient C R	0.1553	0.0656	
Uncertainty Coefficient R C	0.1547	0.0654	
Uncertainty Coefficient Symmetric	0.1550	0.0655	
Sample Size = 48			

Fig. 2.36. SAS Example B9: Measures of association: Fuel and Population

Similar to Pearson's correlation coefficient, the value of Spearman's ρ lies in the range -1 and $+1$, with these values indicating perfect negative or positive association, respectively. For example, if the ranks of one variable agrees perfectly with the ranks of the other variable, $\rho = +1$. Just as with Pearson's correlation coefficient, it is possible to conduct tests based on the t -statistic

$$t = \hat{\rho} \sqrt{\frac{n-2}{1-\hat{\rho}^2}}$$

where $\hat{\rho}$ is the sample rank-correlation coefficient, for testing hypotheses about the population rank-correlation coefficient ρ for sample sizes larger than 10. If the percentage of licenses is used to predict the fuel use category of a state, the nominal measure **asymmetric lambda**, $\lambda(R|C)$, can be used to obtain the proportional reduction in error. Here the value is 0.2903, so that that proportional reduction in error is 29% compared to prediction not based on licensing information.

Including **c1** along with **measures** as **tables** statement options will lead to the computation of asymptotic confidence intervals for the measures of association discussed earlier. The default confidence coefficient is .05, which may be changed by including an optional **alpha=** option with the required value. For some of the measures, adding a **test** statement will produce an asymptotic test of whether the measure is equal to zero as well as an asymptotic confidence interval. These association measures are Gamma, Kendall's τ_b , Stuart's

Gamma		Spearman Correlation Coefficient	
Gamma	0.5641	Correlation	0.4640
ASE	0.1272	ASE	0.1090
95% Lower Conf Limit	0.3148	95% Lower Conf Limit	0.2503
95% Upper Conf Limit	0.8134	95% Upper Conf Limit	0.6776
Test of H0: Gamma = 0		Test of H0: Correlation = 0	
ASE under H0	0.1390	ASE under H0	0.1098
Z	4.0587	Z	4.2268
One-sided Pr > Z	<.0001	One-sided Pr > Z	<.0001
Two-sided Pr > Z	<.0001	Two-sided Pr > Z	<.0001
Sample Size = 48			

Fig. 2.37. Result of TEST statement in PROC FREQ

τ_c , Somers' D , and Pearson's and Spearman's ρ . Fig. 2.37 shows the output resulting from including the statement `test gamma scorr;`. In both cases, the null hypotheses are rejected at reasonable α values. Finally, Fig. 2.38 illustrates how the `list` option may be used to format a cross-tabulation as a one-way table.

taxgrp	fuelgrp	Frequency	Percent	Cumulative Frequency	Cumulative Percent
High	Low Fuel Use	10	20.83	10	20.83
High	Medium Fuel Use	10	20.83	20	41.67
High	High Fuel Use	2	4.17	22	45.83
Low	Low Fuel Use	6	12.50	28	58.33
Low	Medium Fuel Use	7	14.58	35	72.92
Low	High Fuel Use	13	27.08	48	100.00

Fig. 2.38. SAS Example B9: Tax group \times Fuel group cross-tabulation as a list

2.3 Some Useful Base SAS Procedures

There are many Base SAS procedures that calculate a variety of statistics as well as others that perform utility functions. Some of these, such as `proc print`, `proc means`, `proc sort`, and `proc format`, were previously discussed or used in SAS Example programs. Some others such as `proc rank`, and `proc corr` will be used in examples to follow in later chapters. In this section, four useful Base SAS procedures will be briefly introduced and their use illustrated through SAS Example programs.

In Section 2.2, the `plots` option used in `proc univariate` produced a histogram (or a stem-and-leaf plot), a box plot, and a normal probability plot in low-resolution (or line-printer) graphics. See the SAS program displayed in Fig. 2.19 for an example of use of this option. Although high-resolution graphics created by SAS/GRAPH programs are preferable for use in presentations or publications, low-resolution graphics produced by some SAS procedures also play a role, for example, in routine exploratory data analysis or as diagnostic tools. In this section, two procedures that provide statements and options for constructing simple low-resolution scatter plots and charts are introduced.

2.3.1 The PLOT procedure

The general structure of a `proc plot` step is

```
PROC PLOT < options > ;
  BY variable(s) ;
  PLOT plot-request(s) </ option(s)>;
```

A number of `proc` statement options are available for `proc plot`. Although some of these are standard options such as the `data=` option for naming the data set to be analyzed, the `nomiss` for excluding observations with missing values, or the `nolegend` option that suppresses the legend that appears on top of the plot by default, a few are more specialized. Some of these special `proc` options are summarized below.

uniform option specifies that the same scale for the two axes be used for multiple plots produced by the use of the `BY` statement.

formchar < (position(s)) > = 'formatting-character(s)' specifies printable characters to be used for drawing the borders (i.e., outlines) and their intersections on a plot. The `position` parameters identify elements in the border to be plotted: 1 and 2 identify vertical and horizontal lines, 7 identifies the intersection of two of those lines, and 3, 5, 9, and 11 identify the corners, respectively. *formatting-character(s)* define a list of characters assigned to each of these positions. By default, the characters |, -, +, -, -, -, and - are assigned to each of the above seven positions, respectively.

hpercent= option is used to list percentage(s) of the available horizontal space to be used for each plot (in case of multiple plots that are not overlaid).

vpercent= option is used to list percentage(s) of the available vertical space to be used for each plot.

The `plot` statements are called the *action* statements in a `proc plot` step. The operand of a `plot` statement consists of *plot-request(s)* followed by options if necessary.

Plot-requests

The plot request(s) in `plot` statements specify the variables to be plotted on the vertical and horizontal axes, respectively and, optionally, the plotting symbol to be used. For example, the statement

```
plot height*weight;
```

requests a plot of the variables `height` by `weight`; that is, `height` appears on the vertical axis and `weight` appears on the horizontal axis. After `proc plot` determines the scales for the two axes, pairs of values of the two variables (as they occur in each observation) are used to determine the coordinates at which the points are plotted using plot symbols. When a plot symbol is not specified, as in the above example, `proc plot` uses the default scheme of sequentially using uppercase roman letters as plot symbols: 'A' to represent a single observation at a point, B to represent two observations at a point etc. Several more general formats for plot requests are available:

```
vertical*horizontal <$ label-variable>
```

```
vertical*horizontal='character' <$ label-variable>
```

```
vertical*horizontal=variable <$ label-variable>
```

All of these forms optionally allow the specification of a *label variable* that is preceded by a dollar sign (\$). This is the name of a variable that contains strings of characters as values to be used for *labeling* the points plotted. For example, the action statement

```
plot height*weight='*';
```

produces a scatter plot consisting of points indicated by '*' symbols; the statement

```
plot height*weight='*' $ name;
```

will identify those points by 'names', the values of the variable `name`, corresponding to each observation in the data set. On the other hand, the statement

```
plot height*weight=sex;
```

results in the symbols 'M' or 'F', the values of the variable `sex`, to be used as the plot symbols. Plot statement options such as `outward=`, `penalties=`, and `placement=` may be used to control the actual placement of the point labels as described below. When plot request(s) contain multiple requests consisting of several variables, SAS provides syntax to abbreviate the requests. For example, the request `(y1-y2)*(x1-x2)` expands to `y1*x1 y1*x2 y2*x1 y2*x2` and the request `(y1-y2):(x1-x2)` expands to `y1*x1 y2*x2`. Other forms are possible; for example, `y*(x1-x3)` implies `y*x1 y*x2 y*x3` and the request `(x1-x3)` by itself is equivalent to `x1*x2 x1*x3 x2*x3`.

SAS Example B10

As in several previous SAS programs, the SAS program shown in Fig. 2.39 accesses the SAS data set named `fueldat` from a library (a folder, in this case) using the two-level name `mylib.fueldat`. The `proc plot` step contains three plot statements that illustrate forms of plot requests discussed earlier. The three resulting graphs are shown in Figs. 2.40, 2.41, and 2.42, respectively.

```
options ls=60 ps=40 nodate pageno=1;
options formchar="|----|+|----+|-/\<>*" ;
libname mylib 'C:\Documents and Settings\...\stat479\';

proc plot data=mylib.fueldat;
  plot fuel*roads;
  plot fuel*roads='+' ;
  plot fuel*roads='*' $ st;
  title 'Output from Sample Plot Statements';
run;
```

Fig. 2.39. SAS Example B10: Program

Some PLOT Statement Options

Many options are available with the `plot` statement to control the graphics that are output by `proc plot`. Whereas some of these are simple options for controlling the axes, plotting reference lines, and producing multiple plots, others are more complicated options that control the placement of labels at plot points. An abbreviated description of these options is as follows:

haxis= and vaxis= options specify the tick-mark values for the horizontal and vertical axes, respectively. For numeric variables, the axis specification is a list of values such as `haxis=5 10 15 20 25 30 35`. An alternate specification of this option is `haxis=5 to 35 by 5`. If the user wants `proc plot` to determine the minimum and maximum values for the tick marks, just `haxis=by 5` will work. For character variables, the possible tick-mark values are specified as character strings (for e.g., `haxis='Kansas' 'Missouri' 'Iowa' 'Illinois' 'Nebraska'`). For other types of data values, such as *date* values, special syntax is available. For example, `haxis='01MAY07'd to '01DEC07'd by month` specifies a sequence of eight dates as tick-mark values. For dates, the increment may be one of the keywords `day`, `week`, `month`, `qtr`, or `year`. Similar syntax is available for *datetime* and *time* data values.

hpos= and vpos= options control the number of print positions to be used for the horizontal and vertical axes. By default, the `linesize=` (or equivalently, `ls=`) and the `pagesize=` (or equivalently, `ps=`) system options are

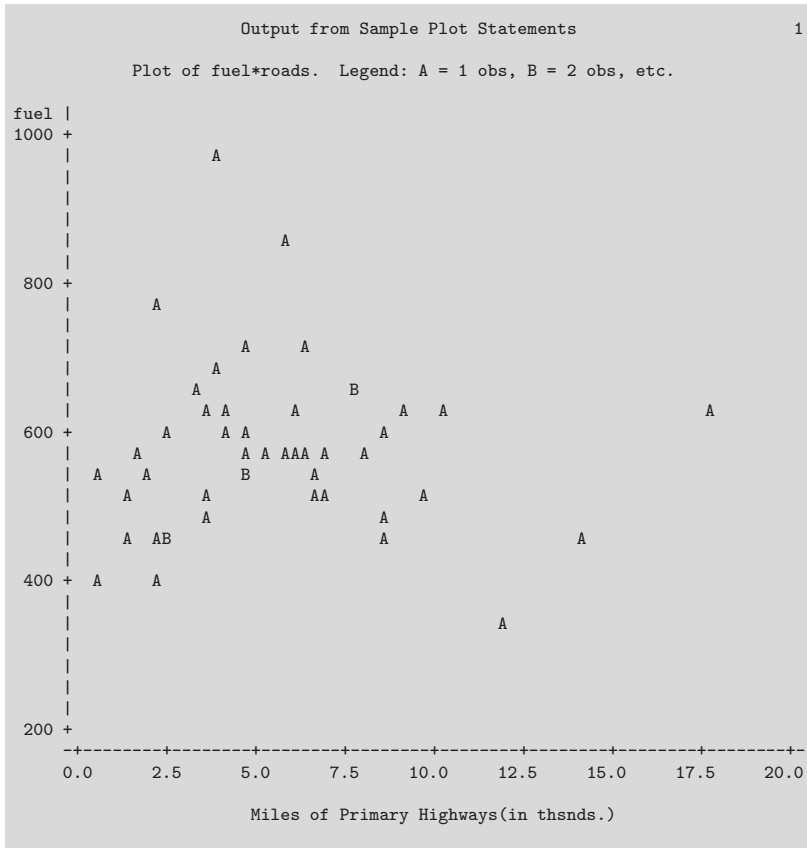


Fig. 2.40. Graph produced by `plot fuel*roads;`

used to determine the horizontal and vertical axis lengths, respectively. Users may change default values set under specific operating environments and displays for these options in an `options` statement.

`href=` and `vref=` options specify the positions on the horizontal and the vertical axis, respectively, at which lines will be drawn on the plot perpendicular to the respective axes. By default, the characters `|` and `-` respectively, will be used to draw the reference lines. The options `hrefchar=` and `vrefchar=` may be used to specify alternate choices (e.g. `hrefchar='.'`).

`box` draws a solid line border around the plot.

`contour <=number-of-levels>` draws a contour plot using plotting symbols to generate degrees of shading where *number-of-levels* is the number of

levels for dividing the range of the ‘third’ variable. The plot request must be of the form **vertical*horizontal=variable** where **variable** contains the “elevation” or “depth” values if the plot is topographical or function values if the plot shows contours of a two-dimensional surface corresponding to a function of two variables.

overlay overlays all plots that are specified in the **plot** statement on one set of axes. The variable names, or variable labels if they exist, from the first plot are used to label the axes. Overlaying is not meaningful unless the sets of variables are comparable, such as the same two variables measured for males and females, or under several different conditions etc. The axes are scaled to fit all sets of variable values unless they are scaled using **haxis=** and/or **vaxis=** options by the user.

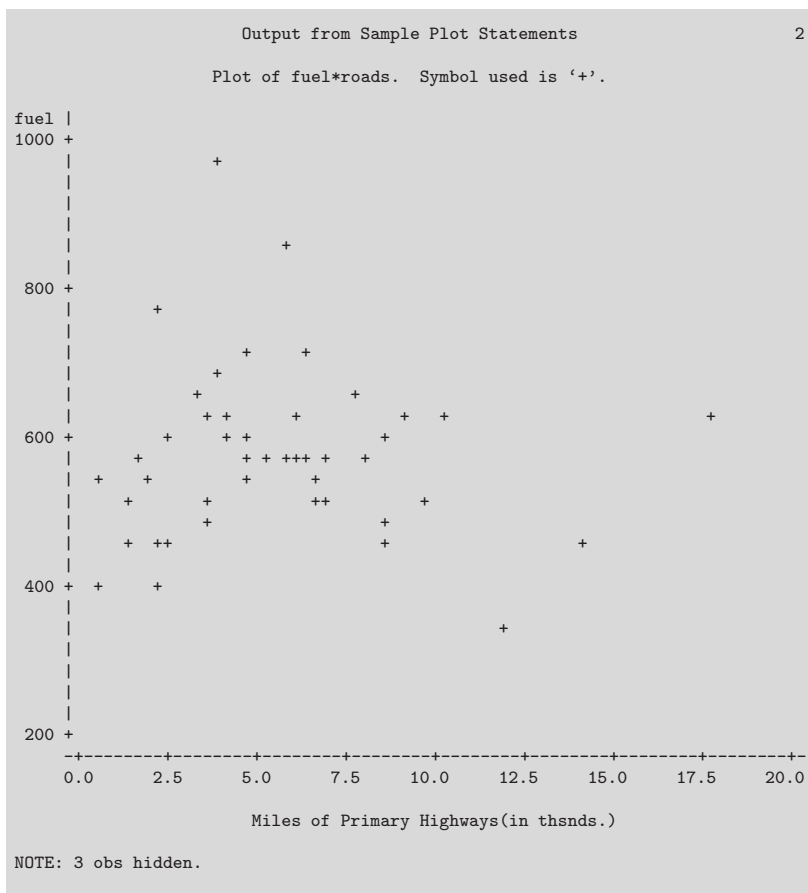
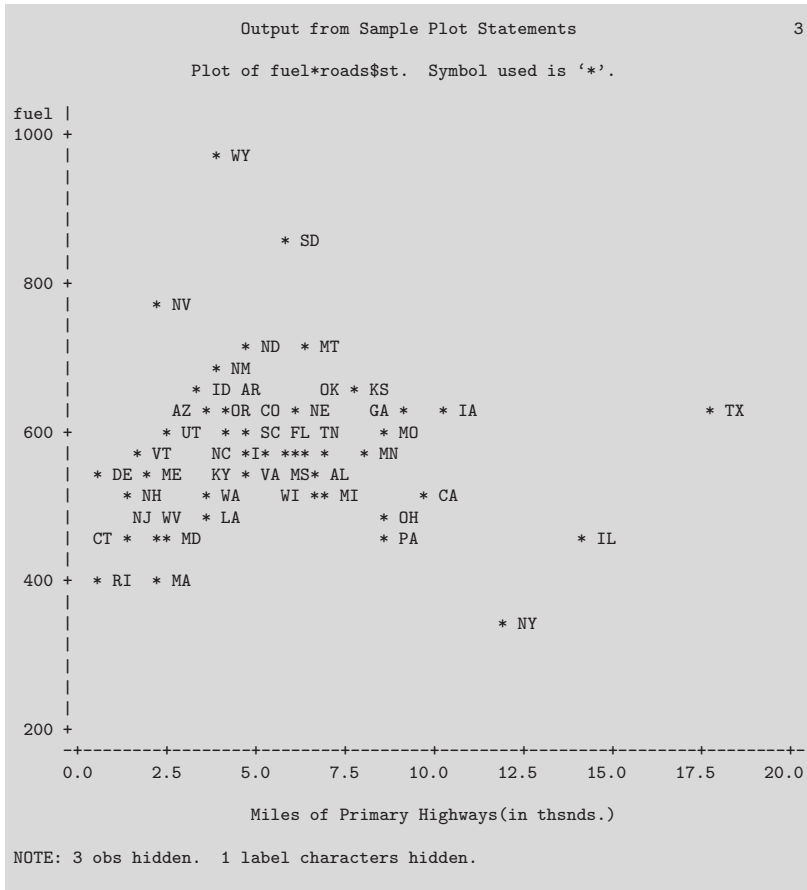


Fig. 2.41. Graph produced by `plot fuel*roads='+';`



<code>penalties< (index-list) >=penalty-list</code>	<code>proc plot</code> determines the best placement positions for labels using a penalization scheme. If every label is placed with zero penalty, then no labels collide and all labels are near their plot symbols. When this is not possible, <code>proc plot</code> determines alternate placements that minimizes the total penalty. To do this, <code>proc plot</code> begins with a <code>default penalty table</code> . For example, a penalty of type 1 (i.e., <code>index-value=1</code> , a default value) is incurred when a nonblank character in the plot collides with an embedded blank in a label or there is not a blank or a plot boundary before or after each (line of the) label. The default penalty value assigned for that event is 1. The option <code>penalties(1)=2</code> would change this penalty to 2, a higher penalty. Generally, several penalty values may be changed for a plot to obtain more preferable placements (e.g., <code>penalties(15 to 19)=2 3 4 10 15 25</code>).
---	---

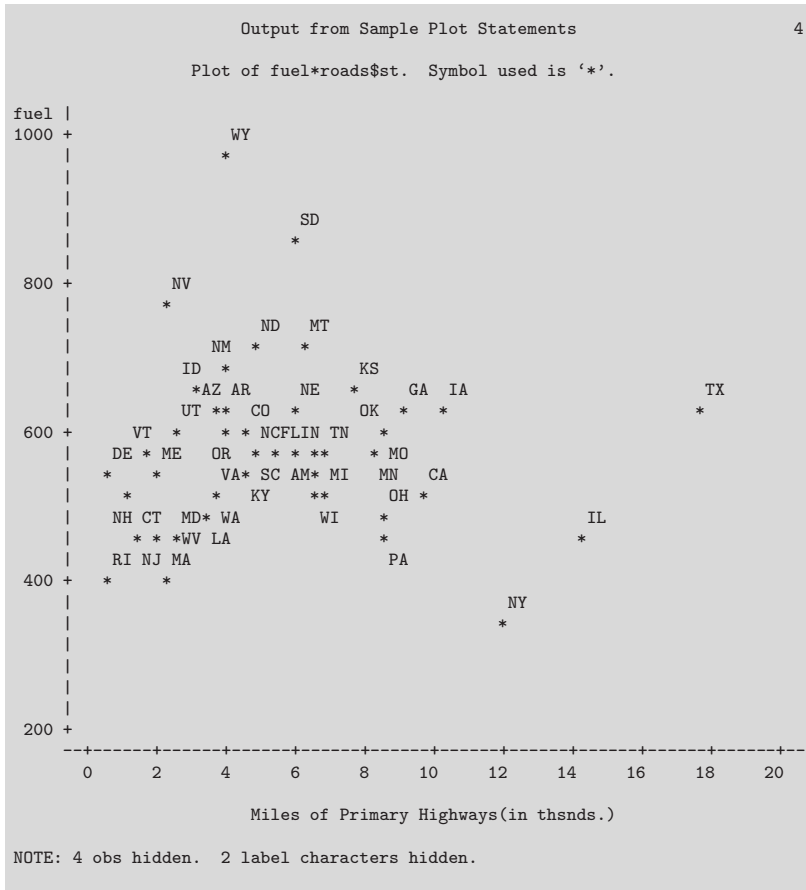
Placement Expressions

The syntax for placement expressions is complicated because it provides a system for generating a large number of expressions by expanding expressions using the symbols `*` and `:` (in the same way as expanding plot requests). A single expression involves setting values for options `h=`, `l=`, `s=`, and `v=`. Each expression is called a *placement state*, which is basically a description of a position where the label can be placed. `proc plot` gives priority to the placement states in the order they occur in a list of placement states. However, it goes through several cycles of refining placement states to find a combination of placements for all labels that minimizes the *total penalty* as well, so the placement states preferred by the user may not always be given priority.

When defining a placement state, the options `h=` and `v=` specify the number of horizontal and vertical spaces to shift the label relative to the starting position, respectively. Both positive and negative integers are allowed, with positive integers signaling shifts to the right or upwards and negative integers indicating shifts to the left or downwards. The option `s=` specifies the starting position, with values `center`, `left`, or `right` indicating whether the string is centered, left-aligned, or right-aligned relative to the plotting symbol location. The option `l=` specifies the number of lines into which the label may be split. If a value is not set for any of these parameters, default values (`h=0`, `v=0`, `s=center`, `l=1`) are assigned.

Multiple placement states can be specified using expressions. For example, specifying `placement=(h=0 1)` results in defining two placement states: (`s=center l=1 h=0 v=0`) and (`s=center l=1 h=1 v=0`). The expression `h=0 1 -1*v=1 -1` results in six different placement states. Other more complex examples are provided in the SAS documentation.

As an example, the following plot statement inserted into the `proc` step in the SAS Example B10 program, displayed Fig. 2.39, produces the graph displayed in Fig. 2.43:



SAS Example B11

This example illustrates the use of the `contour` option in `plot` statements. In addition, it also illustrates the creation of a SAS data set by generating observations entirely with programming statements without reading external data. When input data are not read, a data step goes through just a single iteration. When generating data for low-resolution plots, it is recommended that fewer observations than the number of positions available on the horizontal axis be generated for avoiding overplotting. This rule is not required to be followed when plotting contours, as the objective is to generate a shading using characters that are plotted contiguously; thus overplotting is not a problem.

The program shown in Fig. 2.44 calculates the density function of a bivariate normal distribution $\phi(x, y)$ given by the equation

$$\phi(x, y) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \times \exp \left[\frac{1}{2(1-\rho^2)} \left\{ \left(\frac{x-\mu_1}{\sigma_1} \right)^2 - \frac{2\rho(x-\mu_1)(y-\mu_2)}{\sigma_1\sigma_2} + \left(\frac{y-\mu_2}{\sigma_2} \right)^2 \right\} \right]$$

where random variables X and Y have normal distributions with means μ_1 and μ_2 and variances σ_1^2 and σ_2^2 , respectively, and ρ is the correlation between X and Y . A double `do` loop in the data step calculates the values of the density function $\phi(x, y)$ for values assigned to the variables x and y in the role of

```
data binorm;
mu1=4;sigma1=2;
mu2=10;sigma2=3;
rho=0.4;
  do x=-1 to 9 by .1;
    do y=1 to 18 by .1;
      z1=(x-mu1)/sigma1;
      z2=(y-mu2)/sigma2;
      v=z1**2-2*rho*z1*z2+z2**2;
      const=2*3.14159265*sigma1*sigma2*sqrt(1-rho**2);
      phi=(1/const)*exp(-v/(2*(1-rho**2)));
      output;
    end;
  end;
  drop z1 z2 v const;
run;

proc plot data=binorm nolegend;
  plot y*x=phi / contour=10
        hpos=60 vpos=35
        haxis=-1 to 9 by 2
        vaxis=1 to 18 by 2 ;
  title 'Contour Plot of Bivariate Normal Density';
run;
```

Fig. 2.44. SAS Example B11: Program

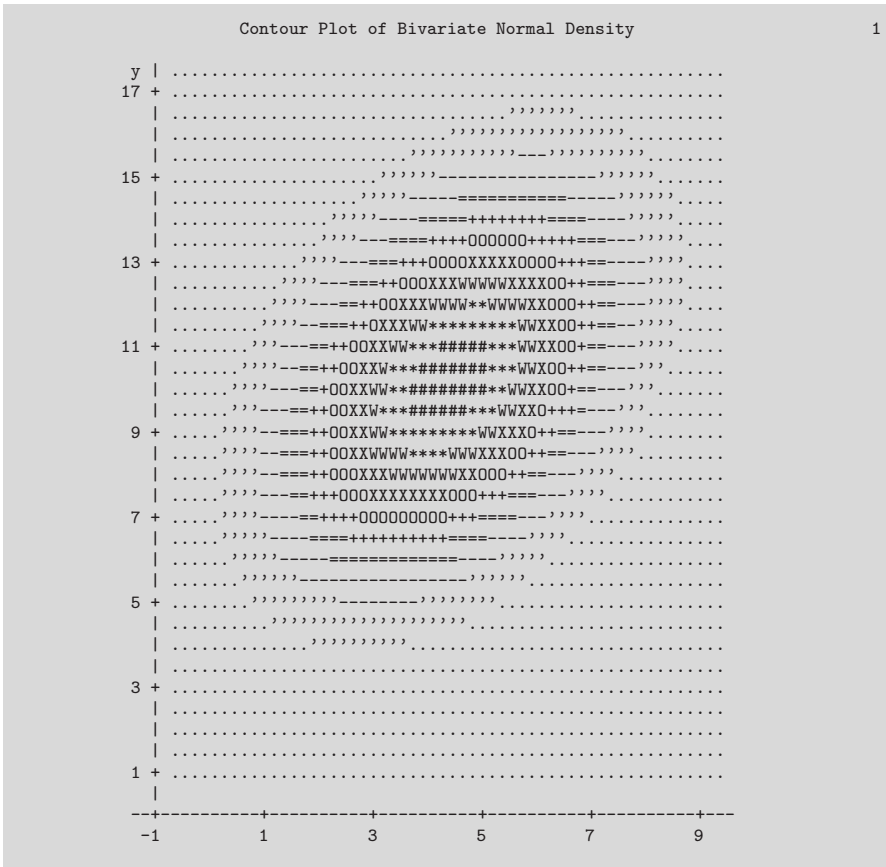


Fig. 2.45. Graph produced by `proc plot` using the `contour=` option

index variables for the two `do` statements. The ranges of values for x and y were determined by considering the possible values for random variables with the specified univariate normal distributions. In each iteration of the double `do loop`, for a pair of values of x and y a value ϕ is calculated and an observation output to the data set. The intermediate variables created were dropped from the data set. A plot of y versus x with ϕ as the third variable and a `contour=` option with 10 levels produced the contour plot shown in Fig. 2.45.

2.3.2 The CHART procedure

The procedure `chart` is typically used to draw low-resolution vertical or horizontal bar charts, block charts, or pie charts. Since the procedure statements in `proc chart` are similar to those available in the SAS/GRAPH procedure

`gchart` discussed in Chapter 3, the discussion in this subsection will be limited to a brief discussion of the statements and some options and providing an example illustrating their use. The general structure of a `proc chart` step (which includes five of the procedure information statements to be illustrated) is

```
PROC CHART < option(s) >;
  BLOCK variable(s) < / option(s)>;
  BY variables;
  HBAR variable(s) < / option(s)>;
  PIE  variable(s) < / option(s)>;
  STAR variable(s) < / option(s)>;
  VBAR variable(s) < / option(s)>;
```

The `proc` statement options in `proc chart` are the `data=` option for naming the data set to be analyzed and the

```
formchar $<($position(s)$)>$ = 'formatting-character(s)'
```

option for specifying printable characters to be used for various elements of a chart. The specifications are different from those used with `proc plot`, but an extensive discussion is not given here. For example, the `position` parameters identify elements in charts: 1 and 2 identify vertical and horizontal axes in bar charts, 7 identifies the tick marks in bar charts, and 9 identifies intersection of axes in bar charts, respectively. *formatting-character(s)* define a list characters assigned to each of these positions. By default, the characters |, -, +, and - are assigned to each of the above four positions, respectively.

The variable(s) that appear in any of the statements `hbar`, `vbar`, `block`, `pie`, or `star` statements above specifies the variable(s) for which these charts are produced. The options that may be specified following the slash in each of these statements enable one to customize the appearance of the charts. By default, if the variable specified is a numeric variable with continuous values, then, by default, the midpoints consist of the midpoints of the class intervals as determined by `proc chart`. For example, in vertical bar charts produced by the `vbar` statement, the midpoint values specify the midpoints of the intervals represented by the bar and these values are plotted at the bottom of each bar. By default, the statistic used to determine the sizes of bars, blocks, or pie sections is the class frequency. For example, in vertical bar charts, class frequency is the variable represented by the vertical axis. In that case the chart produced is a standard histogram. The user may use the option `type=` to specify different choices. For example, in bar charts, `type=percent` requests plotting the percent frequencies.

When the variable specified is discrete-valued such as a character category or a nominal variable or discrete-valued numeric variable such as a level or an ordinal variable, the actual values of the variable determine the placement of the bars, blocks, etc. The option `discrete` must be used to indicate whether the variable is a discrete-valued numeric variable. Otherwise, `proc`

```

libname mylib 'C:\Documents and Settings\...\stat479\';

proc format;
  value ing 1 = 'Low'
           2 = 'Middle'
           3 = 'High';
run;

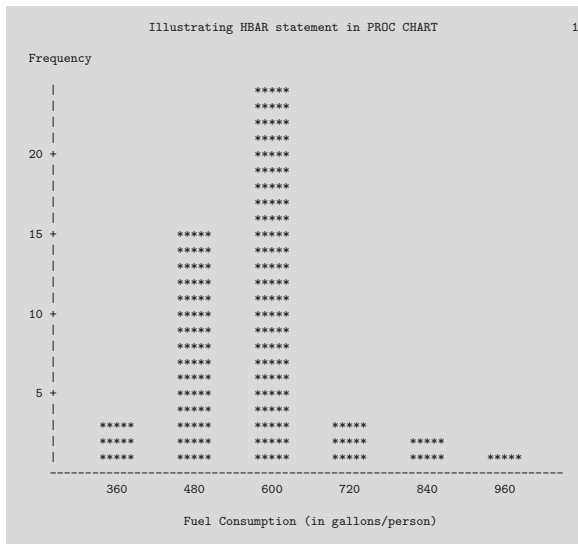
proc chart data=mylib.fueldata;
  vbar fuel;
  vbar fuel/midpoints =300 to 1000 by 100 type=percent;
  vbar incomgrp/discrete sumvar=fuel;
  vbar incomgrp/discrete sumvar=fuel type=mean subgroup=taxgrp;
  vbar incomgrp/discrete sumvar=fuel type=mean group=taxgrp;
  format incomgrp ing.;
  title 'Illustrating HBAR statement in PROC CHART';
run;

```

Fig. 2.46. SAS Example B12: Program

`chart` determines class intervals for the discrete values and incorrectly draws a histogram.

The value represented by each bar, block, etc. are then specified using `sumvar=` and the `type=` options. For example, the `vbar region/sumvar=sales type=mean;` will produce bars representing sales averages from outlets in several regions. Operations needed to produce the bar chart such as the scaling of the vertical axis, determining the bar widths,

Fig. 2.47. Graph produced by `vbar fuel;`

and choosing the spacing between the bars are determined by `proc chart`. However, the user may use options to determine the class intervals and number of bars in a chart, produce side-by-side bars, or subdivide the bars. In some instances, `proc chart` may not accommodate user requests in order to produce a chart. For example, if the number of characters per line available (controlled by the `linesize=` option) is not sufficient to display all of the bars in a request for a vertical bar chart, `proc chart` substitutes a horizontal bar chart.

SAS Example B12

The SAS data set created in SAS Example B6 on fuel consumption data is analyzed in the SAS Example B12 program (see Fig. 2.46) to illustrate the use of `proc chart` for producing low-resolution charts. In this example, the `vbar` statement is selected and the effects of using several options discussed in this section are compared. The two-level name `mylib.fuel` is used to access the previously saved SAS data set named `fuel`, as was done in previous examples. The statement `vbar fuel;` produces the histogram shown in Fig. 2.47. As observed, `proc chart` has selected to draw six bars and placed them at the midpoints 360, 480, 600, 720, 840, and 960. In contrast, as shown in Fig. 2.48, the user opted to use eight bars centered at the midpoints 300, 400, 500, etc. and plot the percent frequency instead. The options needed to achieve this are `midpoints =300 to 1000 by 100` and `type=percent`.

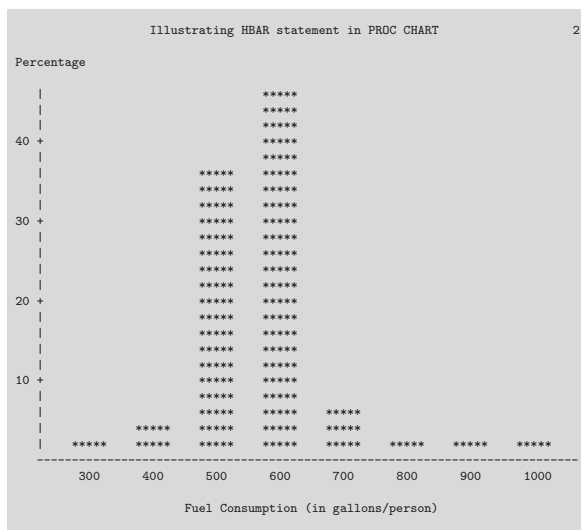


Fig. 2.48. Graph produced by `vbar fuel/midpoints =300 to 1000 by 100 type=percent;`

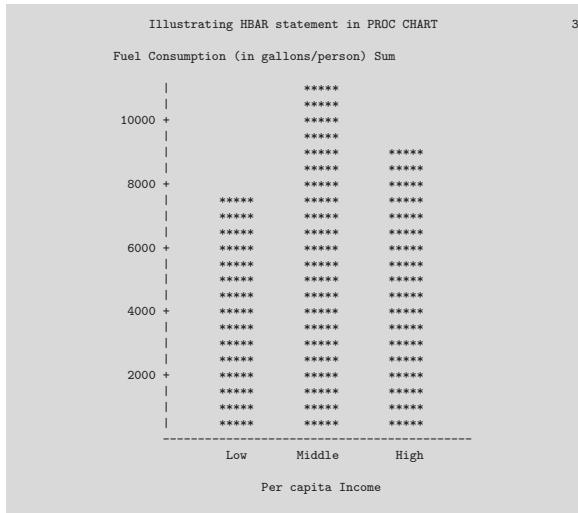


Fig. 2.49. Graph produced by `vbar incomgrp/discrete sumvar=fuel;`

The next `vbar` statement `vbar incomgrp/discrete sumvar=fuel;` uses the category variable `incomgrp` to determine the placement of the vertical bars. This variable is a grouping variable that divides the observations in the data set (i.e., states) into low-, middle-, and high-income groups and has corresponding numeric values of 1, 2, and 3. Thus, the option `discrete` is needed in order to force `proc chart` to regard this variable as a discrete-valued variable. As shown in Fig. 2.49, the variable plotted on the vertical axis is the variable `fuel` as specified in the option `sumvar=fuel` and the size of the bars is determined by default to be the total fuel consumption per person of states (denoted as ‘Sum’ in the plot) in each income group.

The next `vbar` statement also centers the bars at each income group but the option `type=mean` requests that each bar represent the mean fuel consumption for states in each income group rather than the sum, as shown in Fig. 2.50. The option `subgroup=taxgrp` subdivided each bar by the values of the category variable `taxgrp`. Recall that this variable has two values ‘Low’ or ‘High’ depending on whether fuel tax is below 8 cents or not, respectively, for each state. In Fig. 2.50, for example, it can be observed that for middle-income states, the average fuel consumption for low-fuel-tax states is twice as large as those with higher fuel taxes.

The final `vbar` statement produces the plot shown in Fig. 2.51. The `group=taxgrp` option produces side-by-side bar charts for each value of the category variable `taxgrp`. Thus, it produces an analysis of mean fuel consumption for states in each income group within each tax group.

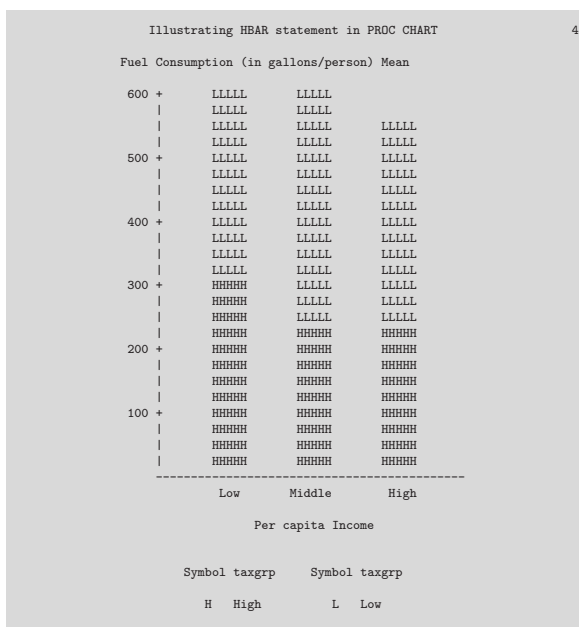


Fig. 2.50. Graph produced by `vbar incomgrp/discrete sumvar=fuel type=mean subgroup=taxgrp`;

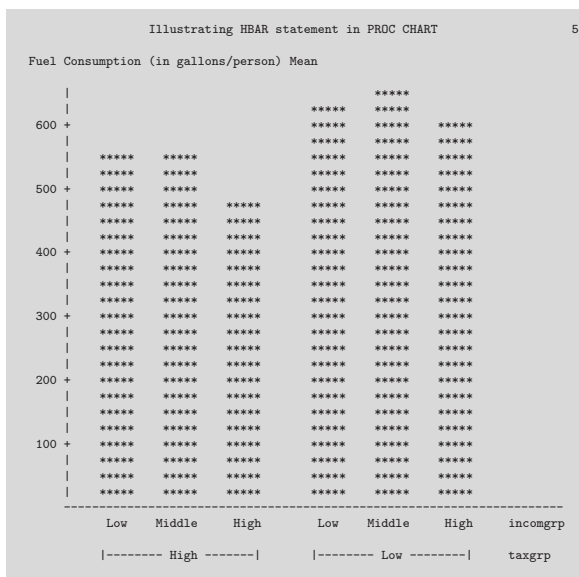


Fig. 2.51. Graph produced by `vbar incomgrp/discrete sumvar=fuel type=mean group=taxgrp`;

2.3.3 The TABULATE procedure

The tabulate procedure is an extremely versatile procedure for producing display-quality tables containing descriptive statistics. Using an extremely simple and flexible system of syntax, the user is able to customize the appearance of the tables incorporating labeling and formatting as well as to generate tabular reports that contain many of the same statistics that are computed by other descriptive statistical procedures. More recent innovations incorporated allow style elements (e.g., colors) to be specified to enhance the appearance of tables output in HTML and RTF formats using ODS graphics output. The statements available in `proc tabulate` and options that can be specified are too numerous to be described in detail in this text. A brief description useful for understanding the example presented below follows. The general structure of a `proc tabulate` step (that excludes several important statements) is

```
PROC TABULATE <options>;
  CLASS variable(s) ;
  VAR   variable(s) ;
  BY    variable(s) ;
  TABLE expression, expression, ... < / options >;
  KEYLABEL keyword='text' ... ;
```

Options available for the `proc` statement are `data=`, `out=`, `missing`, `order=`, `formchar=` (position(s)) `>='formatting-character(s)'`, `noseps`, `alpha=`, `vardef=`, `pctldef=`, `format=`, and `style=`. Since the reader is familiar with many of these options, only those that are relevant to this procedure are described. The default for `formchar=` is `'|---|+|---'` for positions 1 through 11. A value specified for the `format=` option is any valid SAS or user-defined format for printing each cell value in the table, the default being `best12.2`. The `style=` specifies the style element (or style elements) (for the Output Delivery System) to use for each cell of the table. For example, `style=[background=gray]` specifies that the background color for data cells be of gray color. Style elements can be specified in a *dimension expression* (described below) to control the appearance of analysis variable name headings, class variable name headings, class variable level value headings, data cells, keyword headings, and page dimension text.

The `table` statement is the primary statement in `proc tabulate`. The main components of table statements are dimension expressions. A simplified form of the table statement is

```
table expression-1, expression-2, expression-3 </ options> ;
```

where *expression-1* defines the appearance of pages, *expression-2* defines the appearance of rows, and *expression-3* defines the appearance of the columns of the table. A *dimension expression* consists of combinations of the following elements separated by asterisks or blanks:

- Classification variable(s) (variables in the `class` statement)

- Analysis variables (variables in the `var` statement)
- Statistics (`n`, `mean`, `std`, `min`, `max`, etc.)
- Format specifications (e.g., `f=7.2`)
- Nested dimension expressions

If the statements

```
class region popgrp taxgrp;
var fuel income;
```

are present in a `proc tabulate` step, some valid examples of expressions are `region*popgrp`, `region*popgrp*taxgrp`, `region popgrp*taxgrp`, `(region popgrp)* taxgrp`, and `region*mean*fuel`. The expression `region*popgrp` defines a simple two-way table where the row variable is the nominal variable `region` and the column variable is the category variable `popgrp`. Note that the page dimension specification is omitted.

```
libname mylib 'C:\Documents and Settings\...\stat479\';

data fueldat3;
set mylib.fueldat;

if percent=<54 then licgrp=1;
else if 54<percent=<58 then licgrp=2;
else licgrp=3;
run;

proc format;
  value ing 1 = 'Low Income'
           2 = 'Middle Income'
           3 = 'High Income';
  value lg 1='below 54%'
           2='54 to 58%'
           3='above 58%' ;
run;

proc tabulate data=fueldat3;
var fuel;
class incomgrp taxgrp licgrp;
format incomgrp ing. licgrp lg.;
table incomgrp*taxgrp,fuel*(n mean stderr);
table taxgrp*licgrp='Percent Driver Licenses',fuel*(n='Sample Size'*f=4.0
              (mean='Sample Mean' stderr='Standard Error of the Mean')*f=8.1);
title 'Illustrating PROC TABULATE';
run;
```

Fig. 2.52. SAS Example B13: Program

The expression `popgrp*region*taxgrp` produces the same two-way tables for each level of `popgrp` on separate pages. The expression `(region popgrp)* taxgrp`, however, defines a two-way table where the rows are the levels of each of the variables `region` and `popgrp` (not combinations of the levels) and the columns are levels of `taxgrp`. The expression `region*mean*fuel` constructs

a two-way table with the levels of region variable to appear on the rows. The column specification of `mean*fuel` causes the mean of the fuel consumption variable to be computed and appear as a single column. SAS documentation on `proc tabulate` provides ample examples of various combinations used to define dimension expressions; an extensive discussion is omitted here.

SAS Example B13

The SAS data set created in SAS Example B6 on fuel consumption data is used again in the SAS Example B13 program (see Fig. 2.52) to illustrate the use of `proc tabulate` for producing tables of statistics. Recall that `incomgrp` and `taxgrp` are two category variables created previously and included in the data set. In addition, another grouping variable `licgrp` is also created using the values of the variable `percent`, which contains the percentage with driving licenses in the population. From among several analysis variables present in the data set, `fuel` (per capita fuel consumption) is selected for the computation of statistics to be tabulated.

Illustrating PROC TABULATE					1
		Fuel Consumption (in gallons/person)			
		N	Mean	StdErr	
Per capita Income	Fuel Tax				
Low	High	7.00	561.51	18.72	
Income	Low	6.00	626.14	27.27	
Middle	High	7.00	547.93	26.69	
Income	Low	11.00	649.03	35.21	
High	High	8.00	463.14	20.02	
Income	Low	9.00	590.71	49.73	

Fig. 2.53. Output from PROC TABULATE: Page 1

In the SAS program, the `class` statement lists the three category variables and the `var` statement lists the analysis variable. A simple `table` statement is first used to produce a two-way table that tabulates the statistics sample size (n), the mean, and the standard error of the mean (`stderr`) for the variable `fuel`. The first dimension expression `incomgrp*taxgrp` defines the rows of the table, with the rows representing combinations of the levels of `incomgrp` and

Illustrating PROC TABULATE

2

		Fuel Consumption (in gallons/person)		
		Sam- ple Size	Sample Mean	Standard Error of the Mean
Fuel Tax	Percent			
High	Driver Licenses			
	below 54%	8	495.0	26.2
	54 to 58%	10	512.2	19.3
	above 58%	4	597.1	27.5
Low	below 54%	6	552.6	42.3
	54 to 58%	8	591.4	25.7
	above 58%	12	680.5	37.5

Fig. 2.54. Output from PROC TABULATE: Page 2

taxgrp. The second dimension **fuel*(n mean stderr)** defines the columns to be the sample size, the mean, and the standard error of the mean computed for the variable **fuel**. As can be observed, previously defined labels and formats are used for the variables and their levels. The default format of **best12.2** is used for printing all cell values. The default **formchar** specification is used to select characters to draw the table borders, separators, and their intersections if the output is produced for low-resolution printing. This table is shown in Fig. 2.53.

The second **table** statement also produces a two-way table with **taxgrp*licgrp** defining the row and the same statistics computed for the **fuel** variable defining the columns. However, **format** specifications are added to control the formatting of the cell values in the table (e.g., **n*f=4.0**). Also, **label** parameters are added to assign more elaborate labels for class variable names (e.g., **licgrp='Percent Driver Licenses'**) and statistic keyword headings (e.g., **mean='Sample Mean'**). These produce the table shown in Fig. 2.54.

2.4 Exercises

- 2.1 Write a SAS program containing an **infile** statement to access the data set used in SAS Example B5 (see Fig. 2.13) from a text file. Add **proc** step(s) to obtain the following low-resolution plots. Use labels in your data step for the variables and title your plots appropriately.

- a. A scatter plot of weight against height using the data set used in SAS Example B5 (see Fig. 2.13). Use '*' as plot symbols and identify whether each point on your plot represents a male or a female student.
 - b. A vertical bar chart of enrollment in the biology class classified by gender.
 - c. A vertical bar chart of the average height of students classified by gender.
 - d. A horizontal bar chart of the average weight of students classified by year in school within each gender (i.e., side-by-side bar charts for each gender).
 - e. A vertical bar chart (i.e., a histogram) with six bars of the weight of students choosing your own midpoints. Use frequency as the statistic and subdivide bars by gender.
- 2.2 Ott and Longnecker (2001) present an example in which the number of cell clumps per algae species were fitted to a Poisson distribution. A lake sample was analyzed to determine the number of clumps of cells per microscope field. The data are summarized below for 150 fields examined. Here, x_i denotes the number of cell clumps per field and n_i denotes the frequency of occurrence of fields of each cell clump count.

x_i	0	1	2	3	4	5	6	≤ 7
n_i	6	23	29	31	27	13	8	13

Write a SAS program to perform a chi-square goodness-of-fit at $\alpha = .05$ to test the hypothesis that the observed counts were drawn from a Poisson probability distribution.

- 2.3 Devore (1982) discussed an example in which it is examined whether the phenotypes produced from a dihybrid cross of tall cut-leaf tomatoes with dwarf, potato-leaf tomatoes obey the Mendelian laws of inheritance. There are four categories corresponding to the four possible phenotypes: tall cut-leaf, tall potato-leaf, dwarf cut-leaf, and dwarf-potato leaf, with respective expected probabilities p_1, p_2, p_3 , and p_4 . The null hypothesis of interest is

$$H_0 : p_1 = \frac{9}{16}, p_2 = \frac{3}{16}, p_3 = \frac{3}{16}, p_4 = \frac{1}{16}$$

Write a SAS program to perform a chi-square goodness-of-fit at $\alpha = .05$ of this hypothesis given that the observed counts in each category in a sample of size 1611 are 926, 288, 293, and 104, respectively.

- 2.4 The following data, taken from Rice (1988), represent the incidence of tuberculosis in relation to blood groups in a sample of Eskimos. Is there any association of the disease and blood group?

Severity	O	A	AB	B
Moderate-Advanced	7	5	3	13
Minimal	27	32	8	18
Not Present	55	50	7	24

Write a SAS program with a `proc freq` step for performing a chi-square test using $\alpha = .05$ to answer the above question.

- 2.5 Ott et al. (1987) cited a study of migrants to determine whether “degree of kinship participation” in the extended family is independent of a family’s socioeconomic status. Use the data below to compute a chi-square test of independence using `proc freq`. Use nominal measures of association, the contingency coefficient C and Cramer’s V , to comment on the strength of association if present.

Socioeconomic Status	Degree of Kinship Participation		
	Low	Medium	High
Low	75	98	75
Minimal	182	211	123
Not Present	116	122	41

- 2.5 An example in Schlotzhauer and Littell (1997) presented data from an experiment conducted by an epidemiologist who classified the disease severity of dairy cows (none, low, high) by analyzing blood samples for the presence of a bacterial disease. The size of the herd that each cow belonged to was classified as large, medium, or small. One of the aims of this study was to determine if disease severity was affected by herd size.

Severity	Herd Size		
	Large	Medium	Small
None	11	88	136
Low	18	4	19
High	9	5	9

Use a SAS program to analyze these data using `proc freq`. Is there any association between two variables? Use the various measures to interpret any association present considering that the two variables are ordinal and that the experimenter is planning to use herd size for predicting disease severity. Obtain confidence intervals for the measures that you discuss.

- 2.6 Write a SAS program containing an `infile` statement to access the data set used in SAS Example B5 (see Fig. 2.13) from a text file. Use `proc tabulate` to obtain a table laid out as follows. The rows of the table consist of the combinations of year in school and gender, with the levels of gender appearing within each level of year. The column must present mean and standard deviation of the two variables height and weight. Use formats and labels to enhance your table. How can you add a *single* column containing the sample size formatted to print as a four-digit integer?

Exercises 2.7-2.15 concern the demographic data set on countries obtained from Ott et al. (1987) (see Table B.4 of Appendix B). To access this data set from a text file, include an appropriate `infile` statement in your SAS program. A `filename` statement may also be included if you prefer. Use the

following input statement to read these data:

```
input country $20. birthrat deathrat inf_mort life_exp popurban  
      perc_gnp lev_tech civillib;
```

2.7 Write a SAS program to create a SAS data set named `world` using the data in Table B.4. Label variables as appropriate. Create category variables as described below:

Variable	Groupings	Category Variable
Infant mortality	< 24 = 1 (low)	infgrp
	24 – 73 = 2 (moderate)	
	≥ 74 = 3 (high)	
Level of technology	< 24 = 1 (low)	techgrp
	≥ 24 = 2 (high)	
Degree of civil liberties	1, 2 = 1 (low degree of denial)	civilgrp
	3, 4, 5 = 2 (moderate degree of denial)	
	6, 7 = 3 (high degree of denial)	

Use a `libname` statement and a two-level name to save the SAS data set in a folder in your computer.

- 2.8 In a SAS program, use `proc univariate` to compute 33.3 and 66.7 percentiles of the variables `birthrat`, `deathrat`, and `popurban` available in the SAS data set `world`. Access the SAS data set saved previously in Exercise 2.7. Use the `output` statement to save these statistics in a temporary SAS data set named, say, `stats`. Obtain a listing of this data set.
- 2.9 Use the printed output from the analysis performed in Exercise 2.8 to determine good cutoff values for creating additional category variables `birthgrp`, `deathgrp`, and `popgrp` (each with three categories) corresponding to these variables. In a data step of a new SAS program, access the SAS data set `world` saved previously. Add statements to the data step to create the category variables described in Exercise 2.7, name the resulting SAS data set `world2`, and save the new data set in the same folder.
- 2.10 In a new SAS program, use the SAS data set `world2` saved in Exercise 2.9 in a `proc univariate` step to compute descriptive statistics, extreme values, percentiles, and low-resolution plots for the variables `life_exp`, `perc_gnp`, and `popurban`. Use an appropriate option to calculate *t*-tests for the hypotheses that population means for each of these variables exceed 70 years, below \$3000, and above 60%, respectively. Also, include an option for calculating 95% confidence intervals for these parameters. Interpret the results of the *t*-tests using the *p*-values printed. Comment on the shape of the distribution of each of these variables using the printed

output produced. Do the Shapiro-Wilk tests for normality conducted for each variable above support your conclusions?

- 2.11 Add a `proc means` step containing appropriate `class` and `output` statements, to the same SAS program used in Exercise 2.10, to create a SAS data set named `stats1`. The data set `stats1` must contain sample means, standard errors of the means, and maximum and minimum values of the variables `birthrat`, `deathrat`, and `inf_mort` calculated separately for each of the nine groups defined by combinations of levels of the category variables `birthgrp` and `popgrp`. Use the `types` statement to ensure that statistics are calculated only for *combinations of levels* of `birthgrp` and `popgrp`. Suppress printed output in `proc means`. Also, add a `proc format` step to your SAS program to define user formats for use when printing all category variables. Obtain a listing of the data set `stats1`. Label all new variables on output.
- 2.12 Add a `proc plot` step to the SAS program used in Exercise 2.11, to use the SAS data set of sample statistics `stats1` created in the `proc means` step to obtain a low-resolution scatter plot of mean infant mortality versus `popgrp`. The plot symbols must identify the `birthgrp` using formatted values. Recall that a `proc format` was used to define user formats for the `popgrp` and `birthgrp` variables in this program.
- 2.13 In a new SAS program, use the SAS data set `world2` saved in Exercise 2.9 in a `proc freq` step to do the following:
 - a. Obtain two-way frequency tables (in the cross-tabulation format) for the variable `techgrp` with `infgrp`, `techgrp` with `civilgrp`, and `popgrp` with `infgrp`. Compute chi-square statistics, cell χ^2 , and cell expected values but no column, row, or cell percentages,
 - b. Obtain a two-way frequency table (in the list format) for `infgrp` and `civilgrp`.
 - c. Use the chi-square statistic to test hypotheses of independence between pairs of variables considered in part (a) and state your results.
 - d. Use the contingency coefficient C to comment on the strength of association for the pair of variables `techgrp` and `civilgrp`.
 - e. Use the values of Gamma, Kendall's Tau-b, and Spearman correlation coefficient to comment on the association between `techgrp` and `infgrp`.
 - f. Use appropriate measures to evaluate the strength of association between `popgrp` and `infgrp`. Considering that `popgrp` is an independent variable useful for predicting `infgrp` for each country, interpret the appropriate measures of association. Explain.
- 2.14 Write a new SAS program to use the data set (named `world2`) saved in Exercise 2.9 in a `proc chart` step to construct the following low-resolution plots:
 - i. A horizontal bar chart of mean life expectancy for each `infgrp` within each `techgrp` (i.e., side-by-side bar charts for each `techgrp`).

- ii. A vertical bar chart (i.e., a histogram) of per capita GNP, choosing your own midpoints. Use frequency as the statistic and subdivide the bars by `civilgrp`.
- 2.15 Write a SAS program to use the data set (named `world2`) saved in Exercise 2.9, in a `proc tabulate` step to print a tabulation giving the sample size, sample mean, sample standard deviation, and the standard error of the mean of the variables `popurban` for subgroups of observations defined by the combination of values of `techgrp` and `deathgrp`. The row analysis consists of combinations of `techgrp` and `deathgrp` and the statistics for `popurban` must appear on the columns. Print the sample size without decimals, the sample mean with four decimal places, and the other two statistics with two decimal places each. Also, use appropriate text strings to label all statistics keyword headings (e.g., print Standard Deviation for `std`).

SAS for Data Analysis

Intermediate Statistical Methods

Marasinghe, M.G.; Kennedy, W.J.

2008, XII, 558 p. With 100 SAS Programs., Hardcover

ISBN: 978-0-387-77371-1