

7. Class Variables

7.1 Introduction

Survey analysis routinely employs cross-classification. Categorical and crossed variables, described in Chapter 6, offer one means to represent cross-classifications in VPLX. When specific categorical variables, such as geographic areas, age, or other important domains, are intended to cross-classify many other characteristics, then defining these classifiers as CLASS variables provides a generally more powerful approach than crossing. In the unblocked situation described in this chapter, classifying a variable as a class variable globally cross-classifies all remaining variables by the class variable.

This chapter assumes only a reading of Chapters 3 and 4. Chapter 6 is a useful, although not essential, background. On the other hand, this chapter is fundamental for most chapters that follow.

Summary of this chapter:

- Section 7.2 describes the CLASS statement.
- Section 7.3 discusses the role of CLASS variables in the DISPLAY step.
- Section 7.4 provides an example.
- Section 7.5 details how class variables are treated in the CREATE step with respect to features such as COPY, IF blocks, *etc.*

7.2 The CLASS Statement

7.2.1 General Forms. The syntax closely parallels CAT, in Section 6.2. A similar description is repeated here for the sake of those not reading Section 6.2, and to clarify the few differences. The basic syntax of the CLASS statement is

```
CLASS vlist (range1 / range2 /...) ['label1' ['label2'[ ...]]]
```

where *vlist* contains a list of one or more variables, the ranges represent values to be assigned to each of the categories, and, optionally, labels of up to 24 characters for each category are provided. The "/"s in the syntax separate the different categorical levels of variable. For example,

7.2

```
CLASS SEX (1/2) 'Male' 'Female'
```

reclassifies the variable SEX as a class variable by placing values of 1 into the first category and 2 into the second.

Section 3.6.3 describes range specifications. VPLX does not edit the range specifications for overlap. The range specifications are examined in the order specified, and the observation will be classified into the first level in which the specification is satisfied. For example, for

```
CLASS SEX (1 /1-2) 'Male' 'Female'
```

VPLX will classify the value 1 into the first level without noting that the second range is satisfied as well.

Note: Values falling outside of all the specified ranges for a class will have substantially more effect than with CAT. Without blocking, any observation with any of its class variables outside the specified ranges will be entirely excluded from the analysis. With blocking, described in Chapter 8, the observation will not contribute to any block for which one or more class variables are outside the specified ranges. This exclusion is not treated as an error and will not cause the program to terminate.

Although most VPLX CREATE steps establish only a few class variables, the CLASS statement may be used to classify more than one variable at once.

```
class earnings93 earnings94 into any_earn_93 any_earn_94  
  ( 0 /1-high) 'No earnings' 'With earnings'
```

Note that 2 class variables are defined while retaining the original earnings as real variables. The general syntax with INTO takes the form:

```
CLASS vlist1 INTO vlist2 (range1 / range2 ...) ['label1'  
  ['label2'[...]]]
```

where *vlist1* and *vlist2* must have the same length, that is, the same number of variables. New variables appearing in *vlist2* become defined as class variables by this statement.

Because of the strong exclusion effect of class statements, it is often useful to employ *res*, described by Section 3.6.3, to include all remaining values in a residual class. For example,

```
class ethnicity ( 1-8 / res ) 'Hispanic' 'Non-Hispanic'
```

will prevent Non-Hispanics from being dropped from analysis, even if the class variable is set up primarily for the purpose of providing disaggregated estimates for Hispanics but not for non-Hispanics.

When labels are omitted, VPLX will use the specified ranges as labels, that is, "0", "1-9999" etc., in the preceding example.

Single-Level Class Variables: Section 6.2.1 noted that single-level categorical variables are sometimes useful. Single-level class variables, however, function essentially as SELECT statements, and there is consequently no particular advantage to set up single-level class variables.

7.2.2 Labels. The preceding example establishes labels for each category. Note that labels can be separated by commas or blanks. Apostrophes between adjacent labels should not touch, however; like FORTRAN, VPLX translates a pair of adjacent apostrophes into a single apostrophe in the label, as in 'Bachelor' 's Degrees' . Individual labels should be started and completed on the same command line. A command line with an odd number of apostrophes is an error.

7.2.3 Storage of CLASS Variables: At the observation level, class variables are stored in the same manner as categorical variables (Exhibit 6.1). At the level of totals, however, the class variable affects how other variables are stored, including the weighted count of cases.

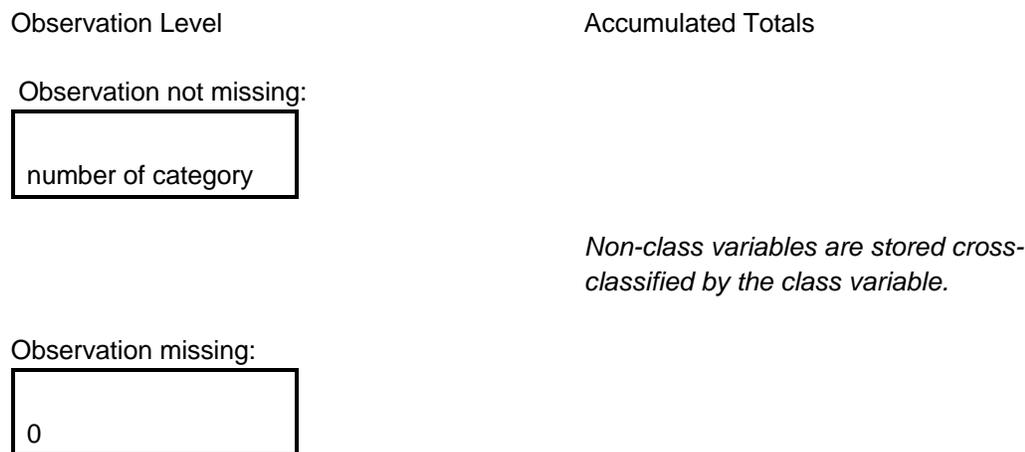


Exhibit 7.1 Representation at the observation and cumulative level for a class variable. The left-hand side shows the observation-level treatment of a class variable. The right-hand side emphasizes that the class variable affects how non-class variables are stored. In a block with classes, totals for all non-class variables are stored completely cross-classified according to the levels of the class variables for

7.4

the block. If the observation does not fall into any level of a class variable, then the observation does not contribute to the totals, even if the remaining variables are otherwise well-defined with valid values.

When one or more CLASS variables are included in the CREATE step to cross-classify data, VPLX tallies only the cross-classification without constructing marginal tables. This strategy requires considerably less storage than one that would construct each possible marginal combination of the class variables as well as their cross-classification. For example, storage for a problem with CLASS variables of sex by a 6-level age categorization requires 12 times the storage without the class variables. If, in addition to the 12 internal cells of cross-classification of the two class variables, VPLX were designed to include all possible margins of the two class variables as well, a multiplier of 21 instead of 12 would be required. For 6 age by 2 sex by 3 race classes, the comparison is between multipliers of 36 and 84. Instead of creating or storing margins in the CREATE step, VPLX offers the ability, through the syntax of the CLASS statement, to create some or all of the margins of the class variables, as needed, within the DISPLAY or TRANSFORM step.

7.3 Treatment of Class Variables in the DISPLAY STEP

Real, real with missing, categorical, and crossed variables, as well as block totals, such as N(1), have all appeared on lists in the DISPLAY step. Class variables are instead used to control the display of other variables, rather than being output directly.

To review, Chapter 4 noted that, except for the statements COMMENT, OPTIONS and SCRATCH1, all other statements in the DISPLAY step are of the form:

request *specification*

where *request* is one of:

LIST - for estimates and standard errors for a list of variables.

COV (or COVARIANCES) - for a variance/covariance matrix for a set of variables.

CORR (or CORRELATIONS) - for a correlation matrix, derived from the covariance matrix, for a set of variables.

T-TEST - for standardized differences, that is, differences divided by estimated standard errors of the differences, for a set of variables. (Section 4.7 illustrates this feature.)

The *specification* is made up of one or more of the following elements, separated by "/":

- 1) **Standard functions of VPLX tallies**, such as means or percentages derived from the file,
- 2) **Class specifications**, to control combinations of class variables, and
- 3) **Options**, to control aspects of the display.

Chapter 4 noted the delay of a description of class specifications until this chapter.

The CLASS specification within a request begins with a "/" to separate it from other parts of the request, followed by "CLASS". Within the DISPLAY step, the specification can list one or more single or multi-dimensional tables. Use of TOTAL implies the total collapsed across all class variables. Multidimensional tables can be built up by using "*" between class names. Class specifications follow the list of variables and functions of variables that they affect. As illustrated previously, there may be more than one class specification in a request.

To illustrate, suppose that three class variables had been established in the CREATE step: sex, age in 6 broad categories, and race in 3 categories.

```
class sex (1/2) 'Males' 'Females'
class race (1/2/res) 'White' 'Black' 'Other races'
class age (16-19/20-29/30-39/40-49/50-64/65-High)
```

In the DISPLAY step, a specification

```
/ class total age race race*sex
```

implies four iterations of the class variables: a display for the total collapsed across age, race, and sex; 6 displays for each of the levels of age, collapsed across sex and race; 3 displays, 1 for each level of race, collapsed across sex and age; and 6 displays for each combination of race and sex, collapsed across age.

It is possible to reference specific levels within a class. For example,

```
/ class age(2,3) race race(2)*sex
```

will produce displays for levels 2 and 3 of age, for all levels of race, and for the second level of race cross-classified by sex.

7.6

It is also possible to combine levels of a class. For example,

```
/ class age(2+3,4,5+6) race(1)*age(2+3,4,5+6)
```

will produce displays for the combination of levels 2 and 3 of age, for level 4 by itself, and for the combination of levels 5 and 6; these age groupings are then shown for the first level of race.

By convention, level 0 of a class represents the total across all levels of the class. Thus, the following 2 specifications have the same effect, except for minor differences in the headings shown in the displays:

```
/ class sex (0-2) * age(0-6)
/ class total, sex, age(1), sex*age(1), age(2), sex*age(2),
      age(3), sex*age(3), age(4), sex*age(4),
      age(5), sex*age(5), age(6), sex*age(6)
```

It is also possible to use *n*, denoting the upper level of the class, as the upper limit on a range, particularly in conjunction with ranges starting with 0. For example, the previous two specifications are equivalent to a third

```
/ class sex (0-n) * age(0-n)
```

Thus, including a class variable in a specification without following it by a range is equivalent to assigning the class variable the range (1-n).

When the CLASS specification is omitted, the results are displayed collapsed across classes. If a variable has not been cross-classified by any classes, then no CLASS specification should be given, or / CLASS TOTAL should be used.

7.4 An Example of CLASS.

The following example is based on an elaboration of exam16.crd and exam18.crd .

```
comment EXAM20

comment This example combines features of EXAM16 and EXAM18
      to illustrate the use of CLASS

create in = exampl11.dat out = exampl11.vpl

input rooms persons cluster tenure / options nprint =3

      4 variables are specified
```

```

format      (4f2.0)

comment    The input data set contains
5 7 1 2
6 8 2 2
5 2 3 1
4 1 4 2
8 4 5 1
8 2 6 1

comment    Make tenure a class variable. Also form a class variable
           from rooms

class      tenure (2 / 1,3 ) 'Renters' 'Owners and others'

class      rooms into rooms_cl (1-5/6-high) '1-5 rooms' '6 or more rooms'

cat        rooms into rooms_cat (1-5/6-high) '1-5 rooms' '6 or more rooms'

cat        persons into persons_cat (1-3/4-high) '1-3 persons'
           '4 or more persons'

labels     rooms 'Number of rooms' persons 'Persons'
           rooms_cat 'Number of rooms' persons_cat 'Number of persons'
           tenure 'Tenure' rooms_cl 'Rooms' rooms_cat 'Rooms'

print     rooms rooms_cl rooms_cat tenure / options nprint = 3

*** PRINT request      1

           (Simple) jackknife replication assumed

           Size of block      1 =                28

           Total size of tally matrix =          28

           Unnamed scratch file opened on unit 13

           Unnamed scratch file opened on unit 14

**** End of CREATE specification/beginning of execution

Observation      1 from unit 12
   rooms          5.0000           persons          7.0000
   cluster        1.0000           tenure          2.0000
PRINT request    1
   rooms          5.000000
   rooms_cl       1.                Class
   rooms_cat      1.                Categorical
   tenure         1.                Class

Observation      2 from unit 12
   rooms          6.0000           persons          8.0000
   cluster        2.0000           tenure          2.0000
PRINT request    1
   rooms          6.000000
   rooms_cl       2.                Class
   rooms_cat      2.                Categorical
   tenure         1.                Class

```

7.8

```

Observation      3 from unit 12
  rooms          5.0000      persons      2.0000
  cluster        3.0000      tenure      1.0000

```

(Printing discontinued on unit 12)

```

PRINT request    1
  rooms          5.000000
  rooms_cl       1.          Class
  rooms_cat      1.          Categorical
  tenure         2.          Class

```

End of primary input file after obs # 6

display

option ndecimal=2 totals

```

list      n(1) rooms rooms_cat persons_cat / class total tenure rooms_cl
          tenure (1) * rooms_cl

```

		Estimate	Standard error
Sample N (wtd) for block	1	6.00	.00
Number of rooms	: TOTAL	36.00	4.10
Rooms	: TOTAL		
1-5 rooms		3.00	1.34
6 or more rooms		3.00	1.34
Number of persons	: TOTAL		
1-3 persons		3.00	1.34
4 or more persons		3.00	1.34
Tenure	: Renters		
		Estimate	Standard error
Sample N (wtd) for block	1	3.00	1.34
Number of rooms	: TOTAL	15.00	6.88
Rooms	: TOTAL		
1-5 rooms		2.00	1.26
6 or more rooms		1.00	1.00
Number of persons	: TOTAL		
1-3 persons		1.00	1.00
4 or more persons		2.00	1.26
Tenure	: Owners and others		
		Estimate	Standard error

Sample N (wtd) for block	1	3.00	1.34
Number of rooms	: TOTAL	21.00	9.77
Rooms	: TOTAL		
1-5 rooms		1.00	1.00
6 or more rooms		2.00	1.26
Number of persons	: TOTAL		
1-3 persons		2.00	1.26
4 or more persons		1.00	1.00

Rooms : 1-5 rooms

Estimate Standard error

Sample N (wtd) for block	1	3.00	1.34
Number of rooms	: TOTAL	14.00	6.32
Rooms	: TOTAL		
1-5 rooms		3.00	1.34
6 or more rooms		.00	.00
Number of persons	: TOTAL		
1-3 persons		2.00	1.26
4 or more persons		1.00	1.00

Rooms : 6 or more rooms

Estimate Standard error

Sample N (wtd) for block	1	3.00	1.34
Number of rooms	: TOTAL	22.00	10.00
Rooms	: TOTAL		
1-5 rooms		.00	.00
6 or more rooms		3.00	1.34
Number of persons	: TOTAL		
1-3 persons		1.00	1.00
4 or more persons		2.00	1.26

Tenure : Renters
Rooms : 1-5 rooms

Estimate Standard error

Sample N (wtd) for block	1	2.00	1.26
Number of rooms	: TOTAL	9.00	5.74

7.10

Rooms	:	TOTAL		
1-5 rooms			2.00	1.26
6 or more rooms			.00	.00
Number of persons	:	TOTAL		
1-3 persons			1.00	1.00
4 or more persons			1.00	1.00
Tenure	:	Renters		
Rooms	:	6 or more rooms		
			Estimate	Standard error
Sample N (wtd) for block	1		1.00	1.00
Number of rooms	:	TOTAL	6.00	6.00
Rooms	:	TOTAL		
1-5 rooms			.00	.00
6 or more rooms			1.00	1.00
Number of persons	:	TOTAL		
1-3 persons			.00	.00
4 or more persons			1.00	1.00

Exhibit 7.2 An example illustrating CLASS in the CREATE step. TENURE and ROOMS_CL are established as class variables. The PRINT statement shows how TENURE stores the level of the variable after the corresponding CLASS statement. In the DISPLAY step, the / CLASS specification results in several different displays, each headed by a description of the levels of the class variables.

7.5 Treatment of Class Variables in the CREATE Step

7.5.1 COPY. If a class variable is copied into a target variable, the target assumes all values and attributes of the class variable, including variable and level labels.

7.5.2 IF Blocks. A class variable may be compared to a range:

```
IF   vname ( range ) THEN
```

As Exhibit 7.2 illustrates, however, the value stored for a class variable is the number of the level to which it belongs, or 0 if it fell outside the available categories. The original value of the categorical variable is, in effect, lost in carrying out the CLASS statement. VPLX will implement an IF statement involving a comparison of a class variable to a range, but prints a warning message. Such comparisons are an easy source of misunderstanding. For example,

```
class age (0-14/15-19/20-34/35-49/50-64/65-high)
if age ( 65 - high ) then
```

The condition will never be satisfied, since, after the CLASS statement, the possible values for age will be 0-6. Instead,

```
class age (0-14/15-19/20-34/35-49/50-64/65-high)
if age ( 6 ) then
```

correctly begins an IF block for persons age 65 and over.

CLASS statements cannot be placed within an IF block. For problems in which it might appear attractive to do so, the alternative is to use IF blocks and other techniques for real variables to produce a real value that can be used in a single CLASS statement.

NOTES
