**SAS PROC TABULATE**

STAT/BIST 5225

Haim Bar

Due by April 6, 2023

On your GitHub Repository

In addition to PROC FREQ, SAS also has PROC TABULATE to create tables. In this class assignment you will learn how to use this procedure by running the examples, and documenting what each program is doing.

**Run the following programs (and if necessary fix bugs), and paste the SAS output after each program. Add a clear description of what the code is doing.**

The BLOOD SALES and MISSING data sets which are needed for this assignment are available on HuskyCT.

The first program uses PROC TABULATE with just one class variable, and all the default options:

proc tabulate data=learn.blood;

class Gender;

table Gender;

run;

The data counts the number of observations for each Gender. There are 440 females and 560 males in the blood dataset.

Table

Description automatically generated

PROC TABULATE has three operators. The first one is “**concatenation**”, which is used by placing two variables in the same TABLE statement, separated by a space:

proc tabulate data=learn.blood format=6.;

class Gender BloodType;

table Gender BloodType;

run;

Table

Description automatically generated

This code joins 2 tables together: on the left side is the data split by Gender (Male and Female), and the left side is the data split by Blood Type (A, AB, B, O).

The following program demonstrates **table dimensions**. Note that there is a very small difference between this program, and the previous one. Explain the difference in terms of the syntax, and in terms of the output.

proc tabulate data=learn.blood format=6.;

class Gender BloodType;

table Gender, BloodType;

run;

This table is different because it lists of male and female by blood type by moving gender to the y axis and blood type on the x axis. This creates a 2 x 4 table of all possible gender-bloodtype combinations, with the observation counts listed in the boxes.

A picture containing table

Description automatically generated

The next program is again very similar to the previous two. Explain what is the difference in terms of the syntax and in terms of the output it produces. This is called **nesting**. Explain why you think it is called nesting.

proc tabulate data=learn.blood format=6.;

class Gender BloodType;

table Gender \* BloodType;

run;

In terms of syntax, the “,” was replaced with a “\*”. This is likely called nesting because blood type is nested under Gender in the table, indicating that first we subset by Gender to get Male and Female and then subset by the four blood types.

Table

Description automatically generated

You can use the keyword ALL in PROC TABULATE by placing it right after the variable name. Of course, it means that you can’t use ALL as a variable name (it’s a reserved word). Explain what adding the ALL keyword does.

proc tabulate data=learn.blood format=6.;

class Gender BloodType;

table Gender ALL,

BloodType ALL;

run;

The ALL keyword creates a sum to add up the counts of all Females and all Males, as in the left part of the table the numbers are broken down by Blood Type

Table

Description automatically generated with low confidence

PROC TABULATE can be used to generate descriptive statistics. The default statistics is the sum. Run the following programs and explain how you can obtain other statistics. Is it possible to make PROC TABULATE generate more than one statistic in a single output? If yes, how?

proc tabulate data=learn.blood;

var RBC WBC;

table RBC WBC;

run;

proc tabulate data=learn.blood;

var RBC WBC;

table RBC\*mean WBC\*mean;

run;

proc tabulate data=learn.blood format=comma9.2;

var RBC WBC;

table (RBC WBC)\*(mean min max);

run;

The first chunk calculates the sum of red and white blood cells. The second chunk calculates the means of red and white blood cells. The third chunk calculate the min, mean, and max of red and white blood cells. Yes it is possible to generate more than one statistic as evidenced by the third chunk. (RBC) \* (mean mode) would calculate the statistics for mean and mode.

Graphical user interface, table

Description automatically generated

The following program uses both categorical and numeric variables (RBC, WBC, and Chol). The \* operator is used twice in the code. Explain what each one is doing.

proc tabulate data=learn.blood format=comma11.2;

class Gender AgeGroup;

var RBC WBC Chol;

table (Gender ALL)\*(AgeGroup All),

(RBC WBC Chol)\*mean;

run;

The first \* is responsible for creating the Gender-AgeGroup part of the table and separating each gender into different age categories. The second \* is responsible for creating the means of Red and White Blood cells and cholesterol for each of the groups generated from the first multiplication sign.

Table

Description automatically generated

Explain what **\*f=...**  does in the following code

proc tabulate data=learn.blood;

var RBC WBC;

table RBC\*mean\*f=7.2 WBC\*mean\*f=comma7.;

run;

\*f appears to be shorthand for format, as when the two \*f are removed 7,043 turns into 7042.97.

Table

Description automatically generated with medium confidence

What does the KEYLABEL statement allows you to do? Try to run the following code with and without it.

proc tabulate data=learn.blood;

class Gender;

var RBC WBC;

table Gender ALL,

RBC\*(mean\*f=9.1 std\*f=9.2)

WBC\*(mean\*f=comma9. std\*f=comma9.1);

keylabel ALL = 'Total'

mean = 'Average'

std = 'Standard Deviation';

run;

Keylabel allows the user to overwrite the names of SAS functions, for example rename Std as “Standard Deviation”

Table

Description automatically generated

You can also customize the output by elimination the N column, if you find it unnecessary:

proc tabulate data=learn.blood format=6.;

class Gender;

table Gender\*n=' ';

run;

This eliminates the N column

Table

Description automatically generated

The following example demonstrates a few additional features. Explain the terms/statements highlighted in red.

proc tabulate data=learn.blood format=comma9.2 noseps;

class Gender AgeGroup;

var RBC WBC Chol;

table (Gender=' ' ALL)\*(AgeGroup=' ' All),

RBC\*(n\*f=3. mean\*f=5.1)

WBC\*(n\*f=3. mean\*f=comma7.)

Chol\*(n\*f=4. mean\*f=7.1);

keylabel ALL = 'Total';

run;

noseps is used to eliminate horizontal separator lines from the row titles and the body of the table

\*f = 5.1 applies a format to the mean, in this case a length of 5 and 1 decimal point. Likewise \*f=comma7. And \*f=7.1 apply formats to WBC mean and Chol mean respectively

Keylabel ALL = “Total” replaces the “All” row/part of table with “Total”

Table

Description automatically generated

The next program shows how to compute percentages in a one-dimensional table. Explain how it is accomplished.

proc tabulate data=learn.blood format=6.;

class BloodType;

table BloodType\*(n pctn);

run;

For each blood type it provides the n (count of # observations) and the pctn which is the #observations/total observations. The \* function applies the methods called after it to the variable called before it.

Table

Description automatically generated

Explain how the following program improves the output of the previous program. The differences are highlighted in red. Explain each one.

proc format;

picture pctfmt low-high='009.9%';

run;

title "Counts and Percentages";

proc tabulate data=learn.blood;

class BloodType;

table (BloodType ALL)\*(n\*f=5. pctn\*f=pctfmt7.1);

keylabel n = 'Count'

pctn = 'Percent';

run;

Table

Description automatically generated

The picture and following format are used to specific that format of the percent variables should contain three digits before the decimal and one afterwards.

The title gives context to the problem

The \*f is used for formatting in this case n is given a length of 5, and pctn is formatted as a percent with a length of 7 and one decimal place

Keylabel renames n and pctn to “Count” and “Percent” to make the code more readable (especially to someone who is unfamiliar with SAS)

Discuss the effect of NOSEPS and /RTS in the following program.

proc format;

picture pctfmt low-high='009.9%';

run;

title "Counts and Percentages";

proc tabulate data=learn.blood noseps;

class Gender BloodType;

table (BloodType ALL),

(Gender ALL)\*(n\*f=5. pctn\*f=pctfmt7.1) /RTS=25;

keylabel ALL = 'Both Genders'

n = 'Count'

pctn = 'Percent';

run;

The “noseps” eliminates horizontal separator lines between rows of output. In this example, it does not do much as there are no horizontal separator lines. Running the code with and without this did not change anything noticeable.

/RTS specifies the maximum number of rows should be displayed in each column before moving to the next column or page. When /RTS is set to be 25, that means a maximum of 25 rows can be displayed before moving on.

Table

Description automatically generated

Explain what COLPCTN does. You may also try ROWPCTN

proc tabulate data=learn.blood noseps;

class Gender BloodType;

table (BloodType ALL='All Blood Types'),

(Gender ALL)\*(n\*f=5. colpctn\*f=pctfmt7.1) /RTS=25;

keylabel All = 'Both Genders'

n = 'Count'

colpctn = 'Percent';

run;

Colpctn calculates the column percentage of observations for each cell (#observations in cell/#column observations) \* 100. Row percent would do something similar but for rows. In the example above, the percent is calculated for each blood type by gender, for example the female by blood type and male by blood type will both add up to 100%.

Table

Description automatically generated

In the following program, notice that we’re using the sales dataset and TotalSales is a numeric variable. What does PCTSUM do?

proc tabulate data=learn.sales;

class Region;

var TotalSales;

table (Region ALL),

TotalSales\*(n\*f=6. sum\*f=dollar8.

pctsum\*f=pctfmt7.);

keylabel ALL = 'All Regions'

n = 'Number of Sales'

sum = 'Average'

pctsum = 'Percent';

label TotalSales = 'Total Sales';

run;

PCTSUM, which is outputted as “Percent” due to the keylabel statement, calculates the percent contribution of each of the regions to the total sales. For example the east region had roughly 41,000 of 77,000 in sales and thus its percent was roughly 54%.

Table

Description automatically generated

For the following programs, use the MISSING data set. Explore the data first, and notice especially the missing data, which affects how PROC TABULATE works.

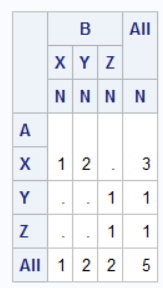
What do you notice when you run the following two programs?

proc tabulate data=learn.missing format=4.;

class A B;

table A ALL,B ALL;

run;



proc tabulate data=learn.missing format=4.;

class A B C;

table A ALL,B ALL;

run;

Application, calendar

Description automatically generated

The only difference between these is that there is no class C in the first table. Both data tables are difficult to interpret due to the missing data.

What happens when you add the MISSING option to PROC TABULATE ?

proc tabulate data=learn.missing format=4. missing;

class A B;

table A ALL,B ALL;

run;

Adding the missing option results in an additional column for the missing values, resulting in there being 4 X values instead of 3 because one of them is counted in the missing column.

Calendar

Description automatically generated

Explain what the MISSTEXT= TABLES option does:

proc tabulate data=learn.missing format=7. missing;

class A B;

table A ALL,B ALL / misstext='no data';

run;

It replaces instances of missing data in the table with text saying “no data”.

Table

Description automatically generated