Post Compute Calculations on IBM SPSS Statistics Pivot Tables

Jon K. Peck
2/20/2023

Table of Contents

Introduction

Most results in SPSS Statistics are displayed in pivot tables. These can be restructured in various ways using the Pivot Table Editor or syntax that manipulates the table. At times, however, cells need to be added to a table that are computed from the table contents. One might, for example, compute an effect size from a table that contains the terms needed but does not display this. The most common usage is to augment the output of the Custom Tables, CTABLES, procedure.

CTABLES allows for post computes via the /PCOMPUTE subcommand or equivalent dialog box. They can refer to cell statistics, including totals and subtotals and combine them using standard arithmetic operators. However, a post compute is limited to computations within the cells of a single variable, and it can only do simple arithmetic.

The STATS TABLE CALC, Utilities > Calculate with Pivot Table, extension command provides a very general post compute capability and is not limited to CTABLES. Except for tables that have layers, computations can be done on almost any table, adding or replacing table cells. The command can even combine results from different tables. This article provides examples of post computes and extends the dialog and syntax help for this procedure. In some cases, the post computes could be created using case variables and modifying the CTABLES or other syntax, but this article focuses only on the post compute technology.

Extension commands work like the built-in commands. In most ways extension commands are indistinguishable from them. Many extension commands are installed with Statistics, but others and updates can be obtained from the Extensions > Extension Hub menu. As some of the examples will illustrate, a bit of Python knowledge can be used to extend the command discussed in this article, but much can be done without that. See the references in the Further Reading section for more information on using Python with SPSS Statistics.

The examples in this article discuss the main points of the syntax, but refer to the syntax (F1) or dialog help for full details.

Note: some of the examples here require at least version 2.0.4 of STATS TABLE CALC. If you have an older version, you can update it to the latest version via Extensions > Extension Hub choosing Updates as the view.

A Simple Comparison of Two Variables

Using CTABLES and a national election survey, this table shows median likes for two variables: Democratic and Republican presidential candidates by occupation of respondent on a scale of 1 to 5. Here is part of the output.

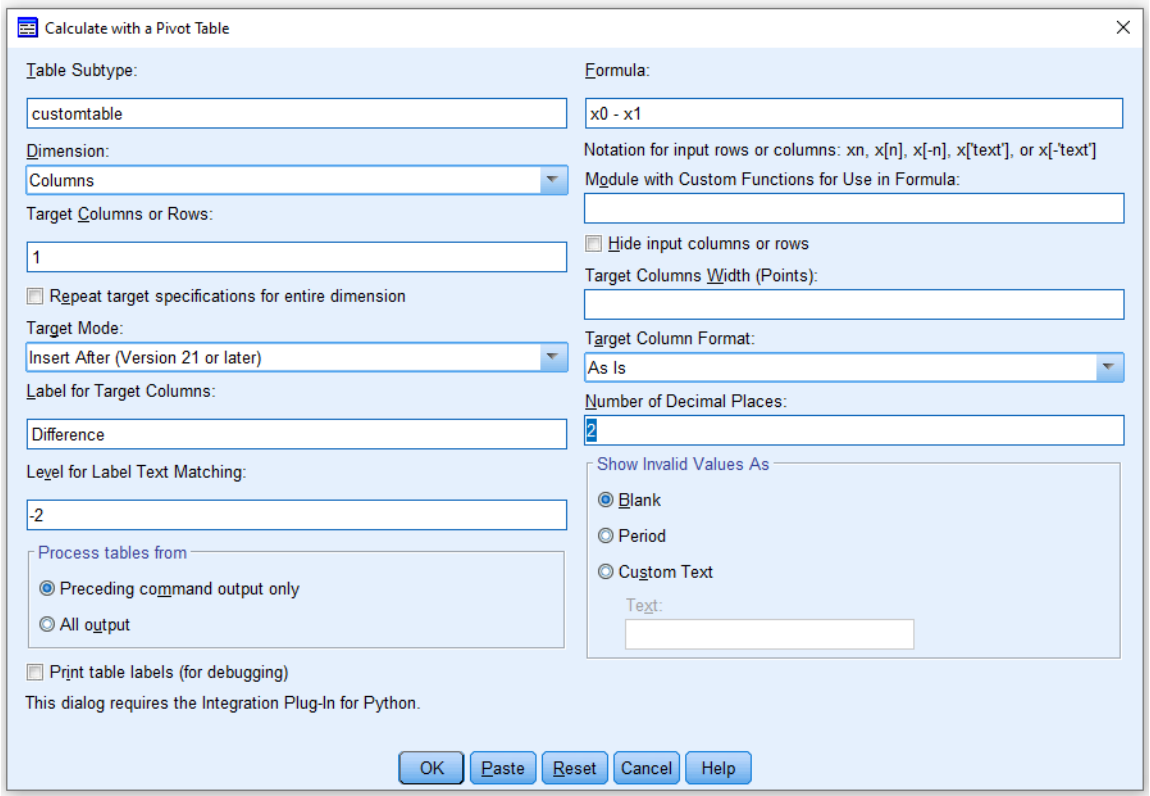
Custom Tables

		likes Dem Pres cand?	likes Rep Pres cand?
		Median	Median
Y16a. OCCNOW		1.00	5.00
	,get prod out on time	5.00	1.00
	,repair of vehicles	5.00	1.00
	... coordinating programs that prevent child abuse	1.00	1.00
	[REDACTED DELIVERY COMPANY NAME] AIR COMPING	1.00	5.00
	[REDACTED DELIVERY COMPANY NAME] driver--local	1.00	1.00
	[REDACTED DETAILS] at a [REDACTED RESTAURANT NAME]	1.00	1.00
	[REDACTED MILITARY OCCUPATIONAL SPECIALTY]	5.00	1.00
	[REDACTED OCCUPATION NAME], fast food services, serving customers, cashier, taking orders, cleaning, preparing food,	1.00	5.00
	[REDACTED PROFESSOR]	1.00	5.00
	[REDACTED RETAIL STORE NAME]	1.00	1.00
	[REDACTED RETAIL STORE NAME] retail sales	1.00	5.00

We would like to include in the table the difference between median likes of both candidates. This can't be done using the CTABLES PCOMPUTE command, because these are different variables, but it can be done using TABLE CALC. Here is the result.

		likes Dem Pres cand?	likes Rep Pres cand?	
		Median	Median	Difference
Y16a. OCCNOW		1.00	5.00	-4.0
	,get prod out on time	5.00	1.00	4.00
	,repair of vehicles	5.00	1.00	4.00
	... coordinating programs that prevent child abuse	1.00	1.00	.00
	[REDACTED DELIVERY COMPANY NAME] AIR COMPING	1.00	5.00	-4.00
	[REDACTED DELIVERY COMPANY NAME] driver--local	1.00	1.00	.00
	[REDACTED DETAILS] at a [REDACTED RESTAURANT NAME]	1.00	1.00	.00
	[REDACTED MILITARY OCCUPATIONAL SPECIALTY]	5.00	1.00	4.00
	[REDACTED OCCUPATION NAME], fast food services, serving customers, cashier, taking orders, cleaning, preparing food,	1.00	5.00	-4.00
	[REDACTED PROFESSOR]	1.00	5.00	-4.00
	[REDACTED RETAIL STORE NAME]	1.00	1.00	.00
	[REDACTED RETAIL STORE	1.00	5.00	4.00

Here are the dialog box and syntax for this.



The syntax as generated by the dialog box is

```
STATS TABLE CALC SUBTYPE="customtable" PROCESS=PRECEDING ❶
/TARGET FORMULA="x0 - x1" ❷
DIMENSION=COLUMNS ❸
LEVEL = -2 ❹
LOCATION=1 ❺
REPEATLOC=NO LABEL="Difference"
MODE=AFTER ❻
HIDEINPUTS=NO
/FORMAT CELLFORMAT="asis" DECIMALS=2  INVALID="".
```

❶ Process the Viewer table with OMS subtype "Customtable" that most closely precedes this command in the Viewer. Use PROCESS=ALL to process all tables of that type.

❷ The formula says to calculate the difference between the column 0 (the first) and column 1 as the statistic. Symbols of the form *xn* refer to absolute locations. *x0* is the first value. The form *x[n]* refers to relative locations: if positive, the column n cells to the right if DIMENSION is COLUMNS or above if the DIMENSION is ROWS. If n is negative, it refers to the location to the left or below. *x[0]* is the target location itself. Note that the formula is enclosed in double quotes.

❸ The dimension specifies whether to process rows or columns. If the dimension is columns, the selected columns will be processed for each row in the table. If it is rows, the selected rows will be processed for each column.

❹ The level refers to the rows in the column headers. -1 is the bottom row, -2 is the one above it. Positive numbers count down from the top.

❺ This specifies where the inserted value goes. It says the second column and ❻ says to insert it after that

column.

Post Compute with a Condition

Here is a CTABLES table showing health conditions by region, counting only Yes answers.

		cond1	cond2	cond3
		Yes	Yes	Yes
		Count	Count	Count
region	1.00	252	27	16
	2.00	142	12	9
	3.00	165	19	10

The conditions are separate variables, so they can't be combined with a total or a PCOMPUTE subcommand. We want to add a column summing the conditions but only if there are at least 20 cases. This is the resulting table. (The data could be restructured using VARSTOCASES so they could totaled, but that still would not accommodate the condition.)

		cond1	cond2	cond3	Common Conditions (N > 20)
		Yes	Yes	Yes	
		Count	Count	Count	
region	1.00	252	27	16	279
	2.00	142	12	9	142
	3.00	165	19	10	165

This is the TABLE CALC syntax. It calculates the sum over the three conditions for cells with a value > 20.

```
STATS TABLE CALC SUBTYPE="customtable" PROCESS=PRECEDING
/TARGET FORMULA="sum(y for y in (x0,x1,x2) if y > 20)" ❶
DIMENSION=COLUMNS LEVEL = -3  ❷
LOCATION=2 MODE=AFTER REPEATLOC=NO
LABEL="Common Conditions (N > 20)"
/FORMAT CELLFORMAT="#.#" DECIMALS=0  INVALID="". ❸
```

❶ The formula groups the three columns into a list including only values > 20 and sums them. The parentheses around the list of x values are necessary for the grouping. If all the values pass the y > 20 condition, the code would be the equivalent of sum(x0, x1, x2). Otherwise, one or more of the arguments would be omitted.

❷ The label for the new column is the top visible row in the label array. A value of -1 would be the innermost level.

❸ The values are formatted as numbers with zero decimals.

A More Complicated Calculation

This example calculates a confidence interval (which is now possible to do directly in CTABLES) illustrating some additional features.

This is the table.

		country					
		1		2		3	
		Count	Column N %	Count	Column N %	Count	Column N %
brand	101	3	50.0%	2	33.3%	1	16.7%
	102	1	16.7%	3	50.0%	3	50.0%
	103	1	16.7%	0	0.0%	1	16.7%
	104	1	16.7%	1	16.7%	1	16.7%

The TABLE CALC syntax is

```
STATS TABLE CALC SUBTYPE="customtable"
/TARGET FORMULA=
"x[0]+(1.96*math.sqrt((x[0]/100)*(1-(x[0]/100))/x[-1]))*100" ❶
LOCATION="Column N %" ❷
REPEATLOC=YES ❸
LABEL="Upper Conf Int" MODE=AFTER
/FORMAT CELLFORMAT="##.#%" ❹
DECIMALS=2  INVALID="".
```

❶ The formula is actually Python code. It refers to relative locations in the table. x[0] is the cell itself; x[-1] is the cell to its left. The square root function, sqrt, from the Python math module is used. There are many other functions available in that module. log (natural log), exp (exponential), trunc (truncate), fsum (sum), fabs (absolute value) are a few of these. Built-in functions, which can be referred to without a module name, include abs, all, min, max, round, sum, and many others.
❷ The target location is specified by its label instead of the column number. This is case sensitive.
❸ This specifies that all the column with that label should be targets.
❹ The format for the new cells, which will be to the right of the "Column N %" cells, is percent with two decimals. The formats are what the Pivot Table Editor shows under "Cell Format".

This is the resulting table. (CTABLES uses a better formula and would give somewhat different results.)

		country								
		1			2			3		
		Count	Column N %	Upper Conf Int	Count	Column N %	Upper Conf Int	Count	Column N %	Upper Conf Int
brand	101	3	50.0%	106.58%	2	33.3%	98.67%	1	16.7%	89.71%
	102	1	16.7%	89.71%	3	50.0%	106.58%	3	50.0%	106.58%
	103	1	16.7%	89.71%	0	0.0%		1	16.7%	89.71%
	104	1	16.7%	89.71%	1	16.7%	89.71%	1	16.7%	89.71%

Going Beyond Simple Formulas

While typical calculations can be done with simple formulas such as these, it is also possible to use a small Python function written by the user for these calculations. In particular, the notation for referring to absolute or relative values in the table is restricted to cells in the same row or column as the target cell. Using Python code, any value in the table can be accessed. The next examples show Python usage.

Calculating Using Other Cells

Starting with this table, which initially shows row percentages,

	DataYear			
	2020		2022	
	Count	Row N %	Count	Row N %
All Members	32,530	49%	34,382	51%
US Members	25,950	49%	26,988	51%
US Specialists	13,615	49%	13,945	51%

Calculate percent of totals in the first row This is not just the column total, because a case could appear in both row 2 and row 3.

First, we define a small Python function named pct in the syntax window.

```
begin program python3.
def pct(arg):    ❶
    '''Calculate percent of total'''

    datacells = arg['datacells'] ❷
    roworcol = arg['roworcol'] ❸
    insertpoint = arg['insertpoint'] ❹


     num = datacells.GetUnformattedValueAt(roworcol, insertpoint-1)❺     denom =
     datacells.GetUnformattedValueAt(0, insertpoint -1) ❻

    return float(num)/float(denom) * 100. ❼

end program.
```

❶ The function is named pct, and has one argument, arg, that contains the information needed from the table. Python code is enclosed within begin program and end program commands. The body of the function must be indented.
❷ datacells is the collection of all the data cells in the table
❸ roworcol is the current cell
❹ insertpoint is the location in the table. These could be accessed directly in the following lines but are extracted separately here for clarify.
❺ Get the unformatted (raw number) value at row roworcol and the column to the left.
❻ Get the unformatted value in the first row and the column to the left.
❼ Calculate and return the percentage.

The command is (SPSS commands are case insensitive except for literal strings)

```
stats table calc subtype="customtable"
/target dimension = columns
label = "All Members %" repeatloc=yes ❶
location="Row N %" level=-1
custommodule="__main__" ❷
formula = "__main__.pct(arg)". ❸
```

❶ Each cell in the column that is labelled at the innermost level "All Members %" is a target cell.
❷ The Python code to be used is found in the syntax window (__main__). If it were in a separate file, this

would be the name of that module. That external module would need to be in a location where Python could find it such as the location where extensions are installed. Run SHOW EXT to see where extensions are installed. ❸ The formula calls the pct function defined above and passes the "arg" structure to it.

This is the resulting table.

	DataYear			
	2017		2020	
	Count	All Members %	Count	All Members %
All Members	32,530	100%	34,382	100%
US Members	25,950	80%	26,988	78%
US Specialists	13,615	42%	13,945	41%

Adding Cohen's d Effect Size Statistic to the T-TEST Output

Effect sizes are now available directly in the T-TEST procedure, but this example shows how to add Cohen's d statistic to one of the T-TEST output tables. Here is a table from T-TEST. Its OMS table type is "Group Statistics".

	Minority Classification	N	Mean	Std. Deviation	Std. Error Mean
Current Salary	No	370	$36,023.31	$18,044.096	$938.068
	Yes	104	$28,713.94	$11,421.638	$1,119.984

The statistics needed for *d* are in this table, but values are needed from both rows, and only one value should be written. Here is the custom function to calculate the value of d. It is saved in a module named cohen.py and is installed with the TABLE CALC extension. (An update to TABLE CALC from the Extension Hub may be needed for this.)

```
import math

def d(arg):
    """Compute Cohen's d statistic from the Group Statistics table
    d is adjusted for bias per Grissom and Kim"""     ❶


    datacells = arg["datacells"]
    roworcol = arg["roworcol"]

    if roworcol > 0:  # Only display in first row
        return ""

    sd1 = float(datacells.GetUnformattedValueAt(0, 2)) ❷
    sd2 = float(datacells.GetUnformattedValueAt(1,2))
    mean1 = float(datacells.GetUnformattedValueAt(0,1))
    mean2 = float(datacells.GetUnformattedValueAt(1,1))
    n1 = float(datacells.GetUnformattedValueAt(0,0))
    n2 = float(datacells.GetUnformattedValueAt(1,0))
    pooledsd = math.sqrt(((n1-1) * sd1**2 + (n2-1) * sd2**2) / (n1+n2))
    d = (mean1 - mean2) / pooledsd  ❸

    # Apply Hedges & Olkin bias adjustment
    dadj = d * (1 - 3 / (4 * (n1 + n2 - 2) - 1))
    return dadj ❹
```

❶ This is called the docstring. It documents the function. Literals enclosed in triple quotes extend over lines until terminated by a matching triple quote.
❷ The necessary statistics are retrieved from the pivot table rows and converted to floating point values for computation.
❸ *d* is calculated from these statistics and the pooled standard deviation.
❹ A bias adjustment is applied, and the value is returned for the first row only.

This is the TABLE CALC syntax.

```
STATS TABLE CALC SUBTYPE="Group Statistics" PROCESS=PRECEDING
/TARGET FORMULA="cohen.d(arg)"  LOCATION=3
 LABEL="Effect Size" MODE=AFTER CUSTOMMODULE="cohen"
/FORMAT DECIMALS=3.
```

The cohen module is in the directory where extensions are installed. This is the updated output.

Group Statistics

	Minority Classification	N	Mean	Std. Deviation	Std. Error Mean	Effect Size
Current Salary	No	370	$36,023.31	$18,044.096	$938.068	0.435
	Yes	104	$28,713.94	$11,421.638	$1,119.984	

Conditional Processing of Table Cells

Here is a table produced by CTABLES using PCOMPUTE subcommands. The syntax is

```
CTABLES
  /PCOMPUTE &cat1 = EXPR(([1] + [2] + [3])/3)
  /PPROPERTIES &cat1 LABEL = "Jan-Mar" FORMAT=COUNT F40.0 HIDESOURCECATS=NO
  /PCOMPUTE &cat2 = EXPR(([2] + [3] + [4])/3)
  /PPROPERTIES &cat2 LABEL = "Feb-Apr" FORMAT=COUNT F40.0 HIDESOURCECATS=NO
  /PCOMPUTE &cat3 = EXPR(([3] + [4] + [5])/3)
  /PPROPERTIES &cat3 LABEL = "Mar-May" FORMAT=COUNT F40.0 HIDESOURCECATS=NO
  /PCOMPUTE &cat4 = EXPR(([4] + [5] + [6])/3)
  /PPROPERTIES &cat4 LABEL = "Apr-Jun" FORMAT=COUNT F40.0 HIDESOURCECATS=NO
  /TABLE Brand > Score [MEAN] BY Month
  /CATEGORIES VARIABLES=Brand ORDER=A KEY=VALUE EMPTY=INCLUDE   /CATEGORIES
VARIABLES=Month [1, 2, 3, 4, 5, 6, &cat1, &cat2, &cat3, &cat4, OTHERNM]
EMPTY=INCLUDE.
```

This is the CTABLES output.

			Month									
			Jan	Feb	Mar	Apr	May	Jun	Jan-Mar	Feb-Apr	Mar-May	Apr-Jun
			Mean	Mean	Mean	Mean	Mean	Mean	Mean	Mean	Mean	Mean
Brand	Brand A	Score	10.02	9.93	10.03	10.08	9.57	10.14	9.99	10.01	9.89	9.93
	Brand B	Score	9.90	.	9.83	9.19	9.02	9.33	.	.	9.34	9.18

The PCOMPUTEs are calculating means across groups of months, e.g., Jan, Feb, Mar, but sometimes data for a month is missing. This causes the resulting calculation to be missing. The requirement is to produce the means based on the months that are not missing. This cannot be done with PCOMPUTE, because its formulas do not

allow for conditional processing. but it can be done with TABLE CALC using, again, a small Python function. (This computed mean is not necessarily the overall mean for the input months as it weights each nonmissing month equally, as do the PCOMPUTES.)

Here is the function. It is run once in a session to make the function available.

begin program python3.

```
import statistics  ❶

def average(arg): ❷
    row = arg['roworcol']
    col = arg['insertpoint']
    datacells = arg['datacells']
    totval =  datacells.GetValueAt(row, col)❸
    if totval != ".": ❹
        return totval

    start = col - 6❺
    items = [] ❻
    for j in range(start, start+3):❼
        val =  datacells.GetValueAt(row, j)❽
        if val != ".": ❾
            items.append(float(datacells.GetUnformattedValueAt(row, j))) ❿
    if items: ⓫
        datacells.SetTextColorAt(row, col, 0xA9A9A9) ⓬
        return statistics.mean(items) ⓭
    else:
        return "." ⓮
end program.
```

❶ Import the statistics module so that its functions can be used.
❷ Define the function named *average* to compute the value. The body of the function must be indented.
❸ Get the formatted cell value at (row, col).
❹ If the value is not missing, i.e., it is not displayed as ".", just return the value.
❺ Look at columns 6 to the left of the current target.
❻ Create an empty list that will contain the cell value.
❼ Loop over 3 cells starting at column *start*.
❽ Get the formatted value at each of those cells.
❾ If the value is not missing…
❿ add the unformatted value to the list. The formatted value would be a string.
⓫ If there are any nonmissing values,
⓬ set the text color for the target cell to gray using RGB code A9A9A9,
⓭ calculate the mean from existing values, and return it.
⓮ Otherwise, return the missing symbol.

RGB codes are readily available on the internet.

Here is the TABLE CALC syntax.

```
STATS TABLE CALC SUBTYPE="Custom Table" PROCESS=PRECEDING
/TARGET CUSTOMMODULE="__main__" FORMULA="__main__.average(arg)"
```

```
DIMENSION=COLUMNS LEVEL = -1
LOCATION=-1 -2 -3 -4 REPEATLOC=NO ❶
LABEL="Mean" MODE=REPLACE
HIDEINPUTS=NO
/FORMAT CELLFORMAT="#.#" ❷
DECIMALS=2  INVALID=".".
```

❶ Set the last four columns in the row as target columns.
❷ Assign a percentage format.

Here is the resulting table. The missing values have been filled in, and values based on incomplete data have been colored light gray.

			Month									
			Jan	Feb	Mar	Apr	May	Jun	Jan-Mar	Feb-Apr	Mar-May	Apr-Jun
			Mean	Mean	Mean	Mean	Mean	Mean	Mean	Mean	Mean	Mean
Brand	Brand A	Score	10.02	9.93	10.03	10.08	9.57	10.14	9.99	10.01	9.89	9.93
	Brand B	Score	9.90	.	9.83	9.19	9.02	9.33	9.86	9.51	9.34	9.18

As this command was executed, these were the argument values for each pf the eight calls. The first column shows the calls going down the rows; the second shows the columns being traversed right to left. The first location, -1, was converted to the column number, so it was passed to the function as 9, and so on.

arg['roworcol']	arg['insertpoint']
0	9
1	9
0	8
1	8
0	7
1	7
0	6
1	6

Modifying a CROSSTABS Table

```
CROSSTABS  /TABLES=jobcat BY gender  /CELLS=COUNT.
```

The CROSSTABS output:

Employment Category * Gender Crosstabulation

Count

		Gender		
		Female	Male	Total
Employment Category	Clerical	206	157	363
	Custodial	0	27	27
	Manager	10	74	84
Total		216	258	474

A custom function in an external module is used to replace the totals with the female percent of the sample in each group.. It could, alternatively, have been defined in the syntax window as above. In the external file, BEGIN/END PROGRAM is not used.

```
def custom(arg, value): ❶
  '''Calculate percent of total'''
    x = float(arg['datacells'].GetUnformattedValueAt(arg['ncells']-1,2)) ❷
    return float(value) * 100 / x
```

❶ An extra parameter containing a value will be passed.
❷ The table total is retrieved for the percentage calculation.

The TABLE CALC command

```
STATS TABLE CALC SUBTYPE='Crosstabulation' ❶
/TARGET DIMENSION=COLUMNS LEVEL=-1 LOCATION="Total" MODE=REPLACE LABEL="Ratio"
CUSTOMMODULE="customexample" ❷
FORMULA =
"customexample.custom(arg, datacells.GetValueAt(roworcol, 0))" ❸
/FORMAT CELLFORMAT="##.#%" DECIMALS=2.
```

❶ Select the table with OMS subtype Crosstabulation.
❷ Use the module named customexample.py containing the required function.
❸ Call the function passing the value at (roworcol, 0).

The result

Count

		Gender		
		Female	Male	Ratio
Employment Category	Clerical	206	157	43.46%
	Custodial	0	27	0.00%
	Manager	10	74	2.11%
Total		216	258	45.57%

This means, for example, that 43.46% of the sample is females in the clerical group, and 45.57% of the entire sample is female.

Using a Secondary Table

It is possible to add results from one or more nearby tables to the main table specified SUBTYPE, but this requires Python code. The keywords NEXTSUBTYPE and PREVSUBTYPE each specify the OMS subtypes to look for going backwards or forwards in the Viewer output relative to the main table.

The syntax (F1) help for TABLE CALC contains examples of the use of the command for combining tables.

More on Python Formulas

arg is a dictionary containing the following items. Passing it as the argument to the function via the formula is the easiest way to expose these properties e.g.,

```
FORMULA = "customexample.custom(arg)"
```

These are the contents of arg.

- pt - the pivot table object
- datacells - the datacells object of the pivot table
- labels - the row or column labels object corresponding to the DIMENSION setting
- ncells - the number of rows or columns in the datacells object. I.e., if DIMENSION is columns, it is the number of rows
- roworcol - the current row or column in the table.
- currentloc - the label of current target column or row in the table
- resultnumber - the number of the value being created
- insertpoints - the row or column numbers for the results before any insertions
- firsttable – a Boolean variable set to True on the first call for a table
- firstload – a Boolean variable set to True on the first call across tables in the command

  firsttable and firstload are provided for use by user functions and have no effect on the operation of TABLE CALC. User code in the function should set these to False as appropriate if using them. Otherwise, they will remain True throughout the command.

The arg dictionary has three additional items that refer to the secondary tables.

- secondary - the secondary pivot table object or objects. If there is more than one, this is a list of objects.
- sdatacells - the datacells object or objects of the secondary pivot table. If there is more than one secondary table, this is a list of objects.
- ncells2 - the number of rows or columns in the secondary datacells object. I.e., if DIMENSION is columns, it is the number or rows. Again, if there is more than one secondary table, this is a list of objects.

  While the contents of arg are fixed, additional named arguments can be specified in the formula and listed in the signature of the function being called, e.g. func(arg, extra=100) would go with a function definition of `def func(arg, extra)`

Further Reading

The Command Syntax Reference manual is available from the Help menu for details on SPSS built-in syntax.

This book (PDF) shows the use of Python code for doing typical SPSS data management tasks. (Only a small part of the book is actually for SAS users.)
SPSS Programming and Data Management
https://www.ibm.com/docs/SSLVMB_29.0.0/pdf/IBM_SPSS_Statistics_Data_Management_SAS.pdf

The Python Reference Guide provides detailed documentation for all of the functions for SPSS.
https://www.ibm.com/docs/SSLVMB_29.0.0/pdf/Python_Reference_Guide_for_IBM_SPSS_Statistics.pdf