Paper 60-27

Anyone Can Learn PROC TABULATE

Lauren Haworth, Genentech, Inc., South San Francisco, CA

➢ ABSTRACT

SAS® Software provides hundreds of ways you can ana-
lyze your data. You can use the DATA step to slice and
dice your data, and there are dozens of procedures that
will process your data and produce all kinds of statistics.
But odds are that no matter how you organize and analyze
your data, you'll end up producing a report in the form of
a table.

This is why every SAS user needs to know how to use
PROC TABULATE. While TABULATE doesn't do any-
thing that you can't do with other PROCs, the payoff is in
the output. TABULATE computes a variety of statistics,
and it neatly packages the results in a single table.

Unfortunately, TABULATE has gotten a bad rap as being
a difficult procedure to learn. This paper will prove that if
you take things step by step, anyone can learn PROC
TABULATE.

This paper is based on Version 8, but all of the examples
save the last one will also work in Version 6.

➢ INTRODUCTION

This paper will start out with the most basic one-
dimensional table. We will then go on to two-dimensional
tables, tables with totals, and finally, three-dimensional
tables. By the end of this paper, you will be ready to build
most basic TABULATE tables.

➢ ONE-DIMENSIONAL TABLES

To really understand TABULATE, you have to start very
simply. The simplest possible table in TABULATE has to
have three things: a PROC TABULATE statement, a
TABLE statement, and a CLASS or VAR statement. In
this example, we will use a VAR statement. Later exam-
ples will show the CLASS statement.

The PROC TABULATE statement looks like this:

```
PROC TABULATE DATA=TEMP;
```

The second part of the procedure is the TABLE statement.
It describes which variables to use and how to arrange the
variables. This first table will have only one variable, so
you don't have to tell TABULATE where to put it. All
you have to do is list it in the TABLE statement. When
there is only one variable, you get a one-dimensional ta-
ble.

```
PROC TABULATE DATA=TEMP;
  TABLE RENT;
RUN;
```

If you run this code as is, you will get an error message
because TABULATE can't figure out whether the vari-
able RENT is intended as an *analysis* variable, which is
used to compute statistics, or a *classification* variable,
which is used to define categories in the table.

In this case, we want to use rent as the analysis variable.
We will be using it to compute a statistic. To tell
TABULATE that RENT is an analysis variable, you use a
VAR statement. The syntax of a VAR statement is sim-
ple: you just list the variables that will be used for analy-
sis. So now the syntax for our PROC TABULATE is:

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  TABLE RENT;
RUN;
```

The result is the table shown below. It has a single col-
umn, with the header RENT to identify the variable, and
the header SUM to identify the statistic. There is just a
single table cell, which contains the value for the sum of
RENT for all of the observations in the dataset TEMP.

Rent
Sum
162898.00

➢ ADDING A STATISTIC

The previous table shows what happens if you don't tell
TABULATE which statistic to use. If the variable in your
table is an analysis variable, meaning that it is listed in a
VAR statement, then the statistic you will get by default
is the sum. Sometimes the sum will be the statistic that
you want. Most likely, sum isn't the statistic that you
want.

To specify the statistic for a PROC TABULATE table,
you modify the TABLE statement. You list the statistic
right after the variable name. To tell TABULATE that the
statistic MEAN should be applied to the variable RENT,
you use an asterisk to link the variable name to the statis-
tic keyword. The asterisk is a TABULATE *operator*. Just
as you use an asterisk as an operator when you want to
multiply 2 by 3 (2*3), you use an asterisk when you want
to apply a statistic to a variable.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  TABLE RENT*MEAN;
RUN;
```

The output with the new statistic is shown below. Note that the variable name at the top of the column heading has remained unchanged. However, the statistic name that is shown in the second line of the heading now says "Mean." In addition, the value shown in the table cell has changed from the sum to the mean.

Rent
Mean
1416.50

➢ ADDING ANOTHER STATISTIC

Each of the tables shown so far was useful, but the power of PROC TABULATE comes from being able to combine several statistics and/or several variables in the same table. TABULATE does this by letting you specify a series of "tables" within a single large table. We're going to add a "table" showing the number of observations to our table showing the mean rent.

The first part of our combined table is the code we used before to compute mean rent.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  TABLE RENT*MEAN;
RUN;
```

Next, we can add similar code to our TABLE statement to get the number of observations. To add this statistic to the first table, all you do is combine the code for the mean ("RENT*MEAN") with the code you would used to get the number of observations ("RENT*N"). The code for the two "tables" is combined by using a space between the two statements. The space operator tells TABULATE that you want to add another column to your table.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  TABLE RENT*N RENT*MEAN;
RUN;
```

The resulting table is shown below.

Rent	Rent
N	Mean
115.00	1416.50

Note that the additional statistic is shown as an additional column in the table. When SAS is creating a one-dimensional table, additional variables, statistics, and categories are always added as new columns.

➢ USING PARENTHESES

While we're just building a simple table, other tables can get complex in a hurry. To keep your table code easy to read, it's helpful to simplify it as much as possible. One thing you can do is use parentheses to avoid repeating elements in row or column definitions.

For example, instead of defining the table statement like this:

```
TABLE RENT*N RENT*MEAN;
```

It can be defined like this:

```
TABLE RENT*(N MEAN);
```

The resulting table is shown below.

Rent	
N	Mean
115.00	1416.50

Note that not only is the table definition easier to read, but the table headings have also been simplified. Now the "Rent" label is not repeated over each column, but rather is listed once as an overall heading. Using parentheses will simplify both your table definitions and your output.

➢ ADDING A CLASSIFICATION VARIABLE

After seeing the tables we've built so far in this chapter, you're probably asking yourself, "Why use PROC TABULATE? Everything I've seen so far could be done with a PROC MEANS."

One answer to this question is classification variables. By specifying a variable to categorize your data, you can produce a concise table that shows values for various subgroups in your data. For example, wouldn't it be more interesting to look at mean rent if it were broken down by city? [1]

To break down rent by city, we will use city as a classification variable. Just as we used a VAR statement to identify our analysis variable, we use a CLASS statement to identify a classification variable. By putting the variable CITY in a CLASS statement, we are telling TABULATE that the variable will be used to identify categories of the data.

The other thing we have to do to our code is tell TABULATE where to put the classification variable CITY in the table. We do this by again using the asterisk operator. By adding another asterisk to the end of the TABLE statement, and following it with the variable

---

[1] The sample data in this paper is from an informal survey of apartment rents in three cities: San Francisco, the author's home town; Orlando, since that's where the paper will be presented; and Seattle, since that's the site of the next SUGI.

name CITY, TABULATE knows that CITY will be used to categorize the mean values of RENT.

```
PROC TABULATE DATA=TEMP;
  CLASS CITY;
  VAR RENT;
  TABLE RENT*MEAN*CITY;
RUN;
```

The resulting table is shown below. Now the column headings have changed. The variable name Rent and the statistic name Mean are still there, but under the statistic label there are now three columns. Each column is headed by the variable label "City" and the category name Portland," "San Francisco," and "Long Beach." The values shown in the table cells now represent subgroup means.

Rent		
Mean		
City		
Orlando	San Francisco	Seattle
837.13	2440.89	1099.55

➢ TWO-DIMENSIONAL TABLES

You probably noticed that our example table is not very elegant in appearance. That's because it only takes advantage of one dimension. It has multiple columns, but only one row. It is much more efficient to build tables that have both rows and columns. You can fit more information on a page, and the table looks better, too.

The easiest way to build a two-dimensional table is to build it one dimension at a time. First, we'll build the columns, and then we'll add the rows.

For this first table, we'll keep things simple. This is the table we built in a previous example. It has two columns: one showing the number of observations for RENT and another showing the mean of RENT.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  TABLE RENT*(N MEAN);
RUN;
```

This table is shown below.

Rent	
N	Mean
115.00	1416.50

To turn this table into a two-dimensional table, we will add another variable to the TABLE statement. In this case, we want to add rows that show the N and MEAN of RENT for different sizes of apartments.

To add another dimension to the table, you use a comma as an operator. All you do is put a comma between the row variable(s) and the column variable(s).

If a TABLE statement has no commas, then it is assumed that the variables and statistics are to be created as columns. If a TABLE statement has two parts, separated by a comma, then TABULATE builds a two-dimensional table using the first part of the TABLE statement as the rows and the second part of the TABLE statement as the columns.

So to get a table with rent as the columns and number of bedrooms as the rows, we just need to add a comma and the variable BEDROOMS. Since we want to add BEDROOMS as a row, we list it before the rest of the TABLE statement. If we wanted to add it as a column, we'd add it to the end of the TABLE statement.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  CLASS BEDROOMS;
  TABLE BEDROOMS, RENT*N RENT*MEAN;
RUN;
```

This table is shown below.

	Rent	Rent
	N	Mean
Bedrooms		
1 Bedroom	57.00	1210.82
2 Bedrooms	58.00	1618.64

By the way, there is a limitation on which variables you can use in a two-dimensional table. You can't have a cross-tabulation of two analysis variables. A two-dimensional table must have at least one classification variable (i.e., you must have a CLASS statement). If you think about it, this makes sense. A table of mean rent by mean bedrooms would be meaningless, but a table of mean rent by categories of bedrooms makes perfect sense.

➢ ADDING CLASSIFICATION VARIABLES ON BOTH DIMENSIONS

The previous example showed how to reformat a table from one to two dimensions, but it did not show the true power of two-dimensional tables. With two dimensions, you can classify your statistics by two different variables at the same time.

To do this, you put one classification variable in the row dimension and one classification in the column dimension. The previous example had bedrooms displayed in rows, and rent as the column variable. In this new table, we will add city as an additional column variable. Instead of just displaying the mean rent for each number of bedrooms, we will display the statistic broken down city.

So in the following code, we leave BEDROOMS as the row variable, and we leave RENT in the column dimension. The only change is to add CITY to the column dimension using the asterisk operator. This tells TABULATE to break down each of the column elements into categories by city.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  CLASS BEDROOMS CITY;
  TABLE BEDROOMS, RENT*CITY*MEAN;
RUN;
```

This table is shown below. Notice how the analysis variable RENT remains as the column heading, and MEAN remains as the statistic, but now there are additional column headings to show the three categories of CITY.

	Rent		
	City		
	Orlando	San Francisco	Seattle
	Mean	Mean	Mean
Bedrooms			
1 Bedroom	726.85	2143.53	902.00
2 Bedrooms	947.40	2721.72	1297.10

➢ ADDING ANOTHER CLASSIFICATION VARIABLE

The previous example showed how to add a classification variable to both the rows and columns of a two-dimensional table. But you are not limited to just one classification per dimension. This next example will show how to display additional subgroups of the data.

In this case, we're going to add availability of high-speed internet access as an additional row classification. The variable is added to the CLASS statement and to the row dimension of the TABLE statement. It is added using a space as the operator, so we will get rows for number of bedrooms followed by rows for internet availability.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  CLASS BEDROOMS CITY INTERNET;
  TABLE BEDROOMS INTERNET,
     RENT*CITY*MEAN;
RUN;
```

In the results shown below, you can see that we now have two two-dimensional "mini-tables" within a single table. First, we have a table of rent by number of bedrooms and city, and then we have a table showing rent by internet availability and city.

	Rent		
	City		
	Orlando	San Francisco	Seattle
	Mean	Mean	Mean
Bedrooms			
1 Bedroom	726.85	2143.53	902.00
2 Bedrooms	947.40	2721.72	1297.10
Internet			
No	769.40	2292.89	855.55
Yes	904.85	2616.63	1343.55

This ability to stack multiple mini-tables within a single table can be a powerful tool for delivering large quantities of information in a user-friendly format.

➢ NESTING THE CLASSIFICATION VARIABLES

So far, all we have done is added additional "tables" to the bottom of our first table. We used the space operator between each of the row variables to produce stacked tables.

By using a series of row variables, we can explore a variety of relationships between the variables. In previous table, we could see how rent varies by number of bedrooms and city, and we could see how rent varies by internet availability and city, but we could not see how the number of bedrooms and internet availability interacted to affect rent for each city.

The power of TABULATE comes from being able to look at combinations of categories within a single table. In the following example, we will build a table to look at rent by city for combinations of number of bedrooms and internet availability.

This code is the same as we used for the last example. The only change is that in the row definition, the asterisk operator is used to show that we want to nest the two row variables. In other words, we want to see the breakdown of rents by internet availability within each category of number of bedrooms.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  CLASS BEDROOMS CITY INTERNET;
  TABLE BEDROOMS*INTERNET,
     RENT*CITY*MEAN;
RUN;
```

As you can see in the table below, this code produces nested categories within the row headings. The row headings are now split into two columns. The first column shows number of bedrooms and the second shows internet availability. It is now easier to interpret the interaction of the two variables.

		Rent		
		City		
		Orlando	San Francisco	Seattle
		Mean	Mean	Mean
Bedrooms	Internet			
1 Bedroom	No	613.80	2012.78	743.30
	Yes	839.90	2290.63	1060.70
2 Bedrooms	No	925.00	2545.00	967.80
	Yes	969.80	2942.63	1626.40

You can also reverse the order of the row variables to look at number of bedrooms by internet availability, instead of internet availability by number of bedrooms. All you do is move INTERNET so that it comes before BEDROOMS. TABULATE always produces the nested rows in the order the variables are listed on the TABLE statement.

➢ ADDING TOTALS TO THE ROWS AND COLUMNS

As your tables get more complex, you can help make your tables more readable by adding row and column totals. Totals are quite easy to generate in TABULATE because you can use the ALL variable. This is a built in classification variable supplied by TABULATE that stands for "all observations." You do not have to list it in the CLASS statement because it is a classification variable by definition.

The following code produces a table similar to the previous example, but with the addition of row totals. The statistic is changed to N so that you can see how the totals work.

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  TABLE BEDROOMS, (CITY ALL)*N;
RUN;
```

As you can see from the table below, the table now has row totals.

	City			
	Orlando	San Francisco	Seattle	All
	N	N	N	N
Bedrooms				
1 Bedroom	20.00	17.00	20.00	57.00
2 Bedrooms	20.00	18.00	20.00	58.00

Not only can you use ALL to add row totals, but you can also use ALL to produce column totals. What you do is list ALL as an additional variable in the row definition of the TABLE statement. No asterisk is needed because we just want to add a total at the bottom of the table.

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  TABLE BEDROOMS ALL, CITY*N;
RUN;
```

The resulting table is shown below. Now there are overall totals for each city.

	City		
	Orlando	San Francisco	Seattle
	N	N	N
Bedrooms			
1 Bedroom	20.00	17.00	20.00
2 Bedrooms	20.00	18.00	20.00
All	40.00	35.00	40.00

➢ THREE-DIMENSIONAL TABLES

Now that you have mastered two-dimensional tables, let's add a third dimension. You may be asking yourself: Three dimensions? How do you print a table shaped like a cube?

Actually, a three-dimensional table is not shaped like a cube. It looks like a two-dimensional table, except that it spans multiple pages. A one-dimensional table just has columns. A two-dimensional table has both columns and rows. A three-dimensional table is just a two-dimensional table that is repeated across multiple pages. You print a new page for each value of the page variable.

The hardest part about three-dimensional tables is making sense of the TABLE statement. So the best way to start is with the first two dimensions: the rows and columns. Once you've got that set up correctly, it's relatively easy to add the page variable to expand the table to multiple pages.

For our example, we're going to build another table of rent by city and number of bedrooms, and then we're going to add internet availability as the page variable. We'll end up with two pages of output, the first page will be for apartments without high-speed internet access, and the second page will be for apartments with high-speed internet access.

Ignoring the third dimension for now, let's build the basic table. This table has rows showing the number of bedrooms, and columns showing rent by city. The code is as follows:

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  VAR RENT;
  TABLE BEDROOMS,
      (CITY ALL)*RENT*MEAN;
RUN;
```

At this point you should run the code and look the table over carefully to be sure you've got exactly what you want to see in your final table.

	City			
	Orlando	San Francisco	Seattle	All
	Rent	Rent	Rent	Rent
	Mean	Mean	Mean	Mean
Bedrooms				
1 Bedroom	726.85	2143.53	902.00	1210.82
2 Bedrooms	947.40	2721.72	1297.10	1618.64

The only difference between this table and our final three-dimensional table is that right now, the table is showing results for both categories of internet access combined. In the final tables, each page will have the results for just one of the two categories.

Assuming the two-dimensional table looks correct, we'll go on to adding the third dimension. When we converted a one-dimensional table to a two-dimensional table, we added the new dimension with a comma operator in the TABLE statement. To change a two-dimensional table to a three-dimensional table, we just add the new variable INTERNET to the existing TABLE statement with a comma to separate it from the row and column definitions.

You might think that the following code is the correct way to add the third dimension to the table statement:

```
TABLE BEDROOMS,
  (CITY ALL)*RENT*MEAN, INTERNET;
```

However, what this would generate is a table with CITY as the rows, INTERNET as the columns, and BEDROOMS as the pages. In order to add the third dimension to a table, you add it at the beginning of the table statements. Remember that this was also true when we added the second dimension to the table. We added rows to the columns by adding a row definition before the column definition.

The correct code for our table is:

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY INTERNET;
  VAR RENT;
  TABLE INTERNET, BEDROOMS,
    (CITY ALL)*RENT*MEAN;
RUN;
```

The resulting tables are shown below. To save space, both tables are shown on a single page. In reality, TABULATE puts a page break between each of the tables, and the table for apartments without high-speed internet access would appear on a second page. Notice that each table was automatically given a title that defines the category it represents.

Internet No

	City			
	Orlando	San Francisco	Seattle	All
	Rent	Rent	Rent	Rent
	Mean	Mean	Mean	Mean
Bedrooms				
1 Bedroom	613.80	2012.78	743.30	1092.62
2 Bedrooms	925.00	2545.00	967.80	1479.27

Internet Yes

	City			
	Orlando	San Francisco	Seattle	All
	Rent	Rent	Rent	Rent
	Mean	Mean	Mean	Mean
Bedrooms				
1 Bedroom	839.90	2290.63	1060.70	1333.25
2 Bedrooms	969.80	2942.63	1626.40	1767.96

➢ MAKING THE TABLE PRETTY

The preceding examples have shown how to create basic tables. They contained all of the needed information, but they were pretty ugly. The next thing you need to learn is a few tricks to clean up your tables.

For example, look at the following code and table:

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  VAR RENT;
  TABLE BEDROOMS,
    (CITY ALL)*RENT*MEAN;
RUN;
```

	City			
	Orlando	San Francisco	Seattle	All
	Rent	Rent	Rent	Rent
	Mean	Mean	Mean	Mean
Bedrooms				
1 Bedroom	726.85	2143.53	902.00	1210.82
2 Bedrooms	947.40	2721.72	1297.10	1618.64

Notice how the totals column is titled "All." We can make this table more readable by changing that title to "Overall."

To do this, we just attach the label to the keyword in the TABLE statement with an equal sign operator. This operator is used to apply labels to variables and statistics. The new code reads as follows:

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  VAR RENT;
  TABLE BEDROOMS,
      (CITY ALL='Overall')*RENT*MEAN;
RUN;
```

This code produces the following output:

	City			
	Orlando	San Francisco	Seattle	Overall
	Rent	Rent	Rent	Rent
	Mean	Mean	Mean	Mean
Bedrooms				
1 Bedroom	726.85	2143.53	902.00	1210.82
2 Bedrooms	947.40	2721.72	1297.10	1618.64

Another thing that could be improved about this table is getting rid of the excessive column headings. This table is four lines deep in headings. For starters, we can get rid of the label "City." With values like "San Francisco," "Portland," and "Long Beach," it's obvious that this table is referring to cities. We don't need the extra label.

To get rid of it, we attach a blank label. A blank label is two quotes with a single blank space in between. The blank label is added the same way we added the "Overall" label in the last example. The blank label is attached to the variable using an equal sign.

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  VAR RENT;
  TABLE BEDROOMS,
      (CITY=' ' ALL='Overall')*
      RENT*MEAN;
RUN;
```

The revised output is shown below.

	Orlando	San Francisco	Seattle	Overall
	Rent	Rent	Rent	Rent
	Mean	Mean	Mean	Mean
Bedrooms				
1 Bedroom	726.85	2143.53	902.00	1210.82
2 Bedrooms	947.40	2721.72	1297.10	1618.64

This looks much better, but there are still too many column headings. We can get rid of two more. Notice how each column is headed by "RENT" and "MEAN."

We can make these labels go away by setting them to a blank label as in the previous example. Also, the row heading "Number of Bedrooms" is not needed, since the categories of "1 Bedroom" and "2 Bedrooms" are self-explanatory. So the variable BEDROOMS is also assigned a blank label. The revised code is shown below.

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  VAR RENT;
  TABLE BEDROOMS=' ',
      (CITY=' ' ALL='Overall')*
      RENT=' '*MEAN=' ';
RUN;
```

Now we have a nice simple table, which is shown below.

	Orlando	San Francisco	Seattle	Overall
1 Bedroom	726.85	2143.53	902.00	1210.82
2 Bedrooms	947.40	2721.72	1297.10	1618.64

However, we also have a problem. If we take the Rent and Mean labels away, then there is no label in the table to describe the analysis variable or statistic.

We could add a title above the table to hold this information, but there's a better way. Notice how in all of our tables there's a big empty box above the rows and to the left of the column headings. This space is available to us. The following code uses the BOX= option to hold a label describing our table.

```
PROC TABULATE DATA=TEMP;
  CLASS BEDROOMS CITY;
  VAR RENT;
  TABLE BEDROOMS,
      (CITY=' ' ALL='Overall')*
      RENT=' '*MEAN=' '
      / BOX='Average Rent';
RUN;
```

The output is shown below:

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	726.85	2143.53	902.00	1210.82
2 Bedrooms	947.40	2721.72	1297.10	1618.64

At this point the table is looking pretty good. There's just one more thing we should do to the table. Since the numbers being reported in the table are rents, it would make the table easier to read if they were formatted with dollar signs. Also, we can round these off to even dollars, that's enough precision for this table.

To change the format of the table cells, we use the FORMAT= option on the TABLE statement. You can use this to apply any valid SAS format. The revised code calls for values to be formatted with dollar signs and commas, and eliminates the display of decimal spaces. The width of 12 was chosen because the widest column label ("Long Beach") is 10 characters wide, and we want an extra space on either side of the label. When you specify a format for the table values, you are also specifying the width of the column that holds that value.

```
    PROC TABULATE DATA=TEMP
        FORMAT=DOLLAR12.;
      CLASS BEDROOMS CITY;
      VAR RENT;
      TABLE BEDROOMS=' ',
          (CITY=' ' ALL='Overall')*
          RENT=' '*MEAN=' '
          / BOX='Average Rent';
    RUN;
```

The revised output is shown below:

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

➢ CREATING HTML OUTPUT

Once you know how to create TABULATE tables, it is a simple matter to turn them into HTML pages that can be posted on your web site.

With versions 8 and higher[1], the process of outputting a TABULATE table to HTML is very simple. All you need is two calls to the Output Delivery System (ODS). The first comes before your TABULATE procedure, and names and opens the output file. The second comes after your TABULATE procedure, and closes the output file.

```
    ODS HTML BODY='SAMPLE.HTML';
    PROC TABULATE DATA=TEMP
        FORMAT=DOLLAR12.;
      CLASS BEDROOMS CITY;
      VAR RENT;
      TABLE BEDROOMS=' ',
          (CITY=' ' ALL='Overall')*
          RENT=' '*MEAN=' '
          / BOX='Average Rent';
    RUN;
    ODS HTML CLOSE;
```

The output is shown below:

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

You can see that the table layout has changed somewhat, and the output now uses a variety of colors and fonts.

➢ CHANGING THE STYLE

With the Output Delivery System, you also have the option of changing the look of your results. By adding a STYLE= option to your ODS statement, you can change from the default style (shown above) to one of the following styles. The resulting output is shown below the code for each style

```
    ODS HTML BODY='SAMPLE.HTML'
        STYLE=XXX;
```

The following examples show a few of the styles that ship with Version 8.

STYLE=BARRETTSBLUE

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

STYLE=BRICK

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

STYLE=BROWN

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

STYLE=D3D

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

STYLE=MINIMAL

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

STYLE=STATDOC

Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

➢ ONE LAST TIP: CREATING A SPREADSHEET

Now that you know how to create HTML output, you already have everything you need to create a spreadsheet from your results. Simply open your HTML file from Excel and your TABULATE output will be imported as a spreadsheet, with all of the rows, columns, and formatting preserved.
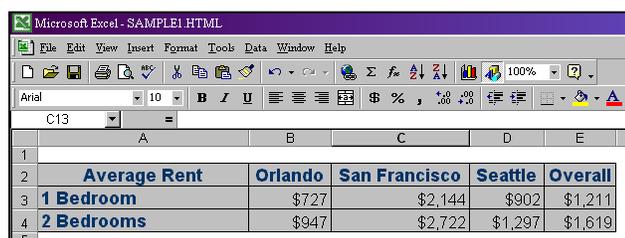
Average Rent	Orlando	San Francisco	Seattle	Overall
1 Bedroom	$727	$2,144	$902	$1,211
2 Bedrooms	$947	$2,722	$1,297	$1,619

If you want only an Excel file, you can even name the file "*filename*.xls" in the ODS HTML statement. Then it will open automatically in Excel when you double-click on the file.

➢ CONCLUSIONS

At this point, you should be comfortable with the basics of producing a table using PROC TABULATE. You should be able to produce a simple table with totals, be able to clean it up a bit, and be able to create HTML output.

This should be enough to get you going producing tables with your own data. And now that you're more comfortable with the procedure, you should be able to use the TABULATE manual and other books and papers to learn more advanced techniques.

➢ ACKNOWLEDGEMENTS

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

➢ CONTACTING THE AUTHOR

Please direct any questions or feedback to the author at:
info@laurenhaworth.com

---

[1] For information on creating HTML output from TABULATE in version 6, see my paper http://www2.sas.com/proceedings/sugi26/p063-26.pdf. It contains instructions for both versions 6 and 8.