

Elegant Tables: Dressing Up Your TABULATE Results

Lauren Haworth
Genentech, Inc., South San Francisco

> INTRODUCTION

Once you've taken the time to learn the basics of TABULATE, you'll quickly discover that creating the table is the easy part. It's making it look nice that's hard.

This paper will show you a number of tips and tricks for designing and formatting your table to make it concise, informative and attractive. The paper then goes on to show how to move your output to HTML and to a word processor so that your results can be easily distributed to others.

> TIP #1: MODIFYING STATISTIC LABELS

This first tip is designed to make your table easier to understand. When we put statistics in our tables, PROC TABULATE adds labels for the appropriate rows or columns that indicate which statistic has been selected. As SAS programmers, the labels N, MEAN, and STD may make perfect sense. However, what about the end-users?

To give them a hand, you can rename the default statistics. The following code shows how it's done. The gray shaded areas are the parts of the code that have been modified to change the labels. To give a statistic a new label to replace the default, place an equal-sign after the statistic keyword, and follow that with the new label in quotes.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  TABLE RENT, ALL*
    (N=Number of Observations,
     MEAN=Average,
     STD=Standard Deviation);
RUN;
```

The resulting table is shown below. Notice that since the new labels were longer than the originals, PROC TABULATE made the spaces for the labels two lines deep. PROC TABULATE will make room for whatever labels you create, but you should try to keep them as short and simple as possible.

	All		
	Number of Observations	Average	Standard Deviation
rent	126.00	1129.66	544.77

> TIP #2: MODIFYING VARIABLE LABELS

Just as we changed the labels for the statistics in the previous example, we can also change the labels for the variables. In the following code, a variable has been labeled in the TABLE statement by using an equal sign after the variable name and then putting the label in quotation marks.

```
PROC TABULATE DATA=TEMP;
  VAR RENT;
  TABLE RENT=Monthly Rent,
    ALL*(N='Number of Observations'
         MEAN='Average'
         STD='Standard Deviation');
RUN;
```

The new table is shown below.

	All		
	Number of Observations	Average	Standard Deviation
Monthly Rent	126.00	1129.66	544.77

> TIP #3: HIDING STATISTIC LABELS

The next area of cleanup we'll tackle is excessive statistic labels. Every time you display a statistic in a table, by default PROC TABULATE generates a row or column heading for that statistic.

In the example below, we have a table that calculates mean rent by city. Because there are three cities, the label "Mean" is repeated three times.

	city		
	Portland	San Francisco	Seattle
	Mean	Mean	Mean
Monthly Rent	859.67	1691.79	1010.37

This table would be much more attractive if we could get rid of the repeated labels. This can be done by using the row variable label to hold the information about the statistic, and then deleting the statistic label. The revised code is below.

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

The statistic label is deleted by assigning it a blank label, which is created by putting a space between two quotes. This code produces the following table. Now the output is much more attractive, but no meaning has been lost (no pun intended).

	city		
	Portland	San Francisco	Seattle
Average Monthly Rent	859.67	1691.79	1010.37

➤ TIP #4: ANOTHER WAY TO RELOCATE THE STATISTIC LABEL

If you don't want to move your statistic into the row label, or it doesn't easily fit into the row label, there's another place you can put the extra information. If you look at the table above, you can see that there's a big empty box in the top left corner of the label. This is valuable space, and TABULATE lets you use it by adding a BOX= option. The code below illustrates how to use this option.

```
PROC TABULATE DATA=TEMP;
  CLASS CITY;
  VAR RENT;
  TABLE RENT='Monthly Rent',
    CITY*MEAN=
  BOX='Average Monthly Rent';
RUN;
```

The new table looks like this:

Averages	city		
	Portland	San Francisco	Seattle
Monthly Rent	859.67	1691.79	1010.37

➤ TIP #5: HIDING A VARIABLE LABEL

Getting rid of the repeated 'MEAN' labels helped simplify the table. However, in the previous example, there's another superfluous label.

If you look at the column headings of the above table, putting the label 'city' above the three values 'Portland', 'San Francisco', and 'Seattle' is redundant. It's obvious that they are cities. This variable label can be removed the same way the 'MEAN' statistic labels were removed, by assigning a blank label.

```
PROC TABULATE DATA=TEMP;
  CLASS CITY;
  VAR RENT;
  TABLE RENT='Monthly Rent',
    CITY*MEAN=
  / BOX='Averages';
RUN;
```

The new table is shown below.

Averages	Portland	San Francisco	Seattle
Monthly Rent	859.67	1691.79	1010.37

Compare this to the original table at the top left of this page. Notice how much more elegant the table looks, and no information has been lost. This is your goal in creating TABULATE output: to refine and simplify the table as much as possible, to make it easier for the reader to follow.

➤ TIP #6: CLEANING UP THE ROW HEADINGS

The previous examples looked at how to get rid of excess column headings. But you can also have problems with row labels. Consider the following table. It's the same table as the previous examples, but turned on its side.

```

PROC TABULATE DATA=TEMP;
  CLASS CITY;
  VAR RENT;
  TABLE CITY=' '*MEAN=' ',
           RENT='Average Monthly Rent';
RUN;

```

All that has changed is that the row and column dimensions have been reversed. However, if you look at the output below, something is wrong.

	Average Monthly Rent
Portland	859.67
San Francisco	1691.79
Seattle	1010.37

This table has an extra set of blank boxes in the row heading. That's because the label for city has been set to a blank label. In the column heading, the box for a blank label is removed from the table. In a row heading, TABULATE leaves behind the empty box. To fix this, you need use the ROW=FLOAT option, as shown in the code below.

```

PROC TABULATE DATA=TEMP;
  CLASS CITY;
  VAR RENT;
  TABLE CITY=' '*MEAN=' ',
           RENT='Average Monthly Rent'
           / ROW=FLOAT;
RUN;

```

Now the table looks like this:

	Average Monthly Rent
Portland	859.67
San Francisco	1691.79
Seattle	1010.37

The ROW=FLOAT option is a good thing to leave turned on all of the time. It does no harm if you have no blank variable or statistic labels in your row headings, and you will need it whenever you do have blank labels.

➤ TIP #7: MODIFYING ROW HEADING WIDTHS

By default, TABULATE divides the line width between the row headings and the table cells

holding row values. It uses a simple formula, which unfortunately does not account for the width of your row variable labels. So quite often, the row headings have too much or too little space for your labels.

In this example, a narrow line size setting has caused the space available for the row heading to become too small to hold the text of the label without wrapping.

	Portland	San Francisco	Seattle
Average Monthly Rent	859.67	1691.79	1010.37

To provide more space, you can override the default width for the row heading using the RTS option. The code for this is as follows:

```

PROC TABULATE DATA=TEMP;
  CLASS CITY;
  VAR RENT;
  TABLE CITY=' '*MEAN=' ',
           RENT='Average Monthly Rent'
           / ROW=FLOAT RTS=22;
RUN;

```

The RTS setting of 22 was computed by taking the width of the label and adding two (for the borders of the row heading). The revised table is shown below

	Portland	San Francisco	Seattle
Average Monthly Rent	859.67	1691.79	1010.37

➤ TIP #8: MODIFYING COLUMN WIDTHS

To change the width of row headings, you use RTS. Unfortunately, there isn't an equivalent CTS option for columns. Instead, to adjust the width of a column, you modify the format of the data displayed in that column.

By default, TABULATE uses the format BEST12.2 for all table values. This means unless you specify otherwise, every column will be 12 spaces wide. In our sample table, 12 spaces looks okay, but if we wanted to save space, we could reduce this to 9 spaces, which is the width of the longest word in a column label.

To change the format, add a FORMAT= option to the PROC TABULATE statement as in the following code.

```
PROC TABULATE DATA=TEMP FORMAT=9;
  CLASS CITY;
  VAR RENT;
  TABLE CITY=' '*MEAN=' ',
         RENT='Average Monthly Rent'
         / ROW=FLOAT RTS=22;
```

RUN;

The new table is shown below: It's a bit more compact than the original.

	Portland	San Francisco	Seattle
Average Monthly Rent	859.67	1691.79	1010.37

➤ TIP #9: USING APPROPRIATE FORMATS

Using the FORMAT= option allowed us to pick an appropriate column width. However, we haven't yet taken full advantage of this option. This table displays results that are dollar amounts. We can use the FORMAT= option to display the results in a more appropriate format. The DOLLAR format adds "\$" and commas to dollar amounts. In addition, since these are large dollar amounts, we can get rid of the decimal places to make the table easier to read.

```
PROC TABULATE DATA=TEMP FORMAT=FORMAT=9;
  CLASS CITY;
  VAR RENT;
  TABLE CITY=' '*MEAN=' ',
         RENT='Average Monthly Rent'
         / ROW=FLOAT RTS=22;
```

RUN;

The new table is shown below:

	Portland	San Francisco	Seattle
Average Monthly Rent	\$860	\$1,692	\$1,010

➤ TIP #10: RE-ORDERING THE HEADINGS

When you generate a TABULATE table, by default the values of your CLASS variable are listed based on their internal values (their unformatted values). TABULATE gives you options to change this to the order of the data (so you can sort the data into the order that you want) or the order of the formatted values.

However, sometimes the table values and their formatted values are not in the order you want. There's a trick you can use to force your headings into the order you desire. To illustrate

the technique, let's say we want to change the table from the previous examples so that the cities are listed in geographic order from north to south.

To get the table in this order you create a new format with leading blanks added to force the formatted values to sort into the order that you want. Then you use the ORDER=FORMATTED option on your PROC TABULATE statement.

```
PROC FORMAT;
  value cityft 1='Portland'
             2='San Francisco'
             3='Seattle';
```

RUN;

```
PROC TABULATE DATA=TEMP FORMAT=9.
```

```
ORDER=FORMATTED;
  CLASS CITY;
  VAR RENT;
  TABLE RENT='Average Monthly Rent',
         CITY=' '*MEAN=' '
         / RTS=22;
```

RUN;

In this example, since we want Seattle to come first, its format is given two leading blanks. Portland comes next, so it gets one leading blank, and San Francisco is left alone. When the table is created, TABULATE uses these blanks to create the column heading order, but strips off the blanks before displaying the table. The result is that the resulting table is in the correct order but doesn't have any extra spaces.

	Seattle	Portland	San Francisco
Average Monthly Rent	\$1,010	\$860	\$1,692

➤ TIP #11: FORMATTING PERCENTAGES

One of the most powerful parts of PROC TABULATE is the ability to specify complex percentages to display in your table. However, the default formatting of these percentages leaves a little to be desired.

For example, the following code is used to produce the table below. It calls for a table of parking availability by city, with column percentages and totals at the end of each column.

```

PROC TABULATE DATA=TEMP
ORDER=FORMATTED;
CLASS PARKING CITY;
TABLE (PARKING='Has Parking' ALL)
      *PCTN<PARKING ALL>=' ',
      CITY=' '
      / ROW=FLOAT RTS=13;
RUN;

```

	Seattle	Portland	San Francisco
Has Parking			
No	47.37	40.74	29.41
Yes	52.63	59.26	70.59
All	100.00	100.00	100.00

Instead of putting percent signs after the percentages, TABULATE uses its standard BEST12.2 format. You might think that switching to the PERCENT format would solve the problem. However, here's what you get if you assign the format of PERCENT9. in the PROC TABULATE statement:

	Seattle	Portland	San Francisco
Has Parking			
No	4737%	4074%	2941%
Yes	5263%	5926%	7059%
All	10000%	10000%	10000%

What happens is that TABULATE multiplies all calculated percentages by 100 before displaying the results. Unfortunately, the PERCENT format also multiplies values by 100, so what you get is values that have been multiplied by 1000!

To add percent signs to TABULATE percentages, you need to create your own percentage format. You can do this with a PICTURE format.

```

PROC TABULATE DATA=TEMP
ORDER=FORMATTED FORMAT=PERCENT9.
CLASS PARKING CITY;
TABLE (PARKING='Has Parking' ALL)
      *PCTN<PARKING ALL>=' ',
      CITY=' '
      / ROW=FLOAT RTS=13;
RUN;

```

RUN;

The new table now has percent signs, but the values have not been altered.

	Seattle	Portland	San Francisco
Has Parking			
No	47%	40%	29%
Yes	52%	59%	70%
All	100%	100%	100%

➤ TIP #12: ANOTHER WAY TO DISPLAY PERCENTAGES

Sometimes TABULATE gives you more output than you need. For example, in the above table we can see that 52% of the Seattle apartments in our dataset have parking available. We don't really need to know that 47% do not. In fact, with the rounding error in this table, it's somewhat confusing to show the percentages for "Yes" and "No". It would be more useful to display a single percentage that shows what percentage of apartments have parking, and ignore the percentage that do not (since it is implied).

You can do this with a simple DATA step trick. You compute a dummy variable that indicates whether the building has parking. All "Yes" answers are coded to 1; all "No" answers are coded to 0. Then you take the mean of that variable in your TABULATE table, and format it as a percentage.

```

PROC TABULATE DATA=TEMP
ORDER=FORMATTED FORMAT=PERCENT9.
CLASS CITY;
TABLE PARKING
      CITY=' '
      / ROW=FLOAT RTS=16;
RUN;

```

In this example, the variable PARKING happens to already be created as 1=Yes, 0=No, so we don't even need a data step. Also note that we can now use the PERCENT format, because we are calculating the percent ourselves so TABULATE will not multiply it by 100. The new table is shown below.

	Seattle	Portland	San Francisco
% With Parking	53%	59%	71%

This version of table is much smaller and easier to interpret. Keep this trick in mind whenever you are displaying percentages of a dichotomous variable.

➤ TIP #13: FORMATTING MISSING VALUES

No data is ever perfect, so chances are you're going to run into the problem of missing data. TABULATE can't make your missing data go away, but it does give you some control on how it is displayed. For example, look at the following table:

Number of Observations	Seattle	Portland	San Francisco
Has Parking			
No	.	22	10
Yes	20	32	24

This table has a period in one cell where there is missing data. This convention is familiar to SAS programmers, but the end user of your table may not know what the "." stands for.

Since this is a table of Ns, it would be nice to display a zero in that empty cell instead of the period. You can request this with the MISSTEXT option.

```
PROC TABULATE DATA=TEMPMISS
  ORDER=FORMATTED FORMAT=9.;
  CLASS CITY PARKING;
  TABLE PARKING='Has Parking'*N=' ',
    CITY=' ',
    / ROW=FLOAT RTS=15
    BOX='Number of Observations'
    MISSTEXT=0;
RUN;
```

The revised table is displayed below.

Number of Observations	Seattle	Portland	San Francisco
Has Parking			
No	0	22	10
Yes	20	32	24

➤ TIP #14: REMOVING REDUNDANT INFORMATION

This next example uses several tricks to simplify a table with redundant information.

```
PROC TABULATE DATA=TEMP
  ORDER=FORMATTED FORMAT=6.;
  CLASS CITY;
  VAR RENT BEDROOMS;
  TABLE RENT='Monthly Rent'
    BEDROOMS='Number of Bedrooms',
    CITY=' '*(N MEAN)
    / ROW=FLOAT RTS=14.;
RUN;
```

The table is designed to show the N and MEAN for two variables for each city. However, notice that the N is the same for both variables in each case. So space is being wasted displaying the N twice in each column.

	Seattle		Portland		San Francisco	
	N	Mean	N	Mean	N	Mean
Monthly Rent	38	1010	54	860	34	1692
Number of Bedrooms	38	2	54	2	34	2

The table can be simplified by only displaying the N for one of the two variables. It can also be made more user friendly by displaying the results in one column per city instead of two.

```
PROC TABULATE DATA=TEMP
  ORDER=FORMATTED;
  CLASS CITY;
  VAR RENT BEDROOMS;
  TABLE RENT='Average Rent'
    MEAN='Average Bedrooms'
    BEDROOMS='Average Bedrooms'
    CITY=' ',
    / ROW=FLOAT RTS=16.;
RUN;
```

The trick in this table is to move the statistics to the row dimension and to use the variable RENT twice in the table statement. The first time it is labeled as "(N)" and is used to compute the N statistic. The second time it is labeled as "Average Rent" and it is used to compute the MEAN statistic. The second variable BEDROOMS is only used to compute a mean.

Also, notice how formats have been used in the TABLE statement. If you want to use a single format for the whole table, specify it in the PROC TABULATE statement. However, if you want to use different formats for each statistic or variable, you can apply the format directly to the statistic or variable by using an asterisk and the FORMAT= or F= syntax. In this example, F= is used to format the N as an integer, the variable RENT as a dollar amount, and the variable BEDROOMS as a number with two decimal places.

The new table, shown below, now has only three columns (instead of 6) and has one new row.

	Seattle	Portland	San Francisco
(N)	38	54	34
Average Rent	\$1,010	\$860	\$1,692
Average Bedrooms	1.50	1.50	1.50

Whenever your table has repeated information, think of using tricks like this to simplify it.

➤ TIP #15: REMOVING THE ROW DIVIDERS

In each of the examples so far, we've used the default PROC TABULATE table grid. The table grid is the horizontal and vertical bar characters that make up the borders of the table rows, columns, and cells. The default table grid puts a border around every cell, row, column, and page. However, you don't have to use the standard table grid. If you'd like a table that looks a little less "boxy," you can tell PROC TABULATE to remove the row dividers.

Average Rent	Seattle	Portland	San Francisco
1 Bedroom	\$887	\$621	\$1,284
2 Bedrooms	\$1,134	\$1,099	\$2,100

If you look closely at the above table, you will see that each row in the table actually requires

two lines of text. One has the data that goes in each of the table cells. The second line consists of a series of characters that form the divider between the rows. The following code uses the NOSEPS option to remove these row dividers from your table.

```
PROC TABULATE DATA=TEMP
  ORDER=FORMATTED F=DOLLAR9.;
  CLASS CITY BEDROOMS;
  VAR RENT;
  TABLE RENT=' '*BEDROOMS=' ',
    CITY=' '*MEAN=' '
    / ROW=FLOAT RTS=16
    BOX='Average Rent';
RUN;
```

The new table retains the boxes around the column headers, but now there are no row dividers in the body of the table.

Average Rent	Seattle	Portland	San Francisco
1 Bedroom	\$887	\$621	\$1,284
2 Bedrooms	\$1,134	\$1,099	\$2,100

This can be a great space-saving tool if you have a table that is too tall to fit on the page, but be careful. If your table has many columns, it may be hard to see which numbers line up across a row without the help of the row dividers.

➤ TIP #16: MODIFYING THE BORDER STYLE

Throughout this paper, the tables have been displayed with high-resolution smooth lines as borders. However, when you run TABULATE, your tables may look more like this:

Average Rent	Seattle	Portland	San Francisco
1 Bedroom	\$887	\$621	\$1,284
2 Bedrooms	\$1,134	\$1,099	\$2,100

This setup is designed so that it can be displayed on low-resolution monitors, and printed on a line printer. But these days, even if you're on a mainframe, you're probably printing to a high-resolution laser printer. If you'd like to set up TABULATE to print tables with smooth line borders, you need to change your FORMCHAR option setting.

There are two ways to do this. One is to look up the hex codes for each of the special border characters for your printer. This means digging

out the printer manual and finding the codes for characters like “ π ” and “ τ ”.

The other option is to use the following FORMCHAR setting:

```
OPTIONS FORMCHAR='82838485868788898A8B8C'X;
```

If you're running SAS on Windows, you can change your font setting to SAS Monospace, and your table will be displayed and printed with the smooth borders of the previous examples.

If you're running SAS in an environment where you can't change the display and printing font, then you'll need to save your output and open it in a word processor. You can change the font to SAS Monospace and print it from there.

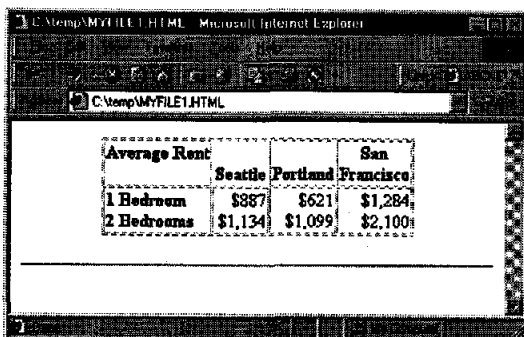
➤ TIP #17: EXPORTING TO HTML

If you plan on displaying your results on the Internet, or sharing them via the company intranet, or e-mailing the results to a client, the best thing you can do to make your output more attractive is to convert it to HTML.

In version 6, the code for creating HTML is:

```
OPTIONS FORMCHAR='82838485868788898A8B8C'X;
%TAB2HTM (CAPTURE=ON, RUNMODE=B);
PROC TABULATE DATA=TEMP NOSEPS
  ORDER=FORMATTED F=DOLLAR9.;
  CLASS CITY BEDROOMS;
  VAR RENT;
  TABLE RENT=' '*BEDROOMS=' ',
    CITY=' '*MEAN=' '
    / ROW=FLOAT RTS=16
    BOX='Average Rent';
RUN;
%TAB2HTM(CAPTURE=OFF, RUNMODE=B,
  OPENMODE=REPLACE,
  HTMLFILE=C:\TEMP\MYFILE1.HTML);
```

This example shows a minimal set of options, there are many more. In particular, note the FORMCHAR setting, which is required. The output is shown below.

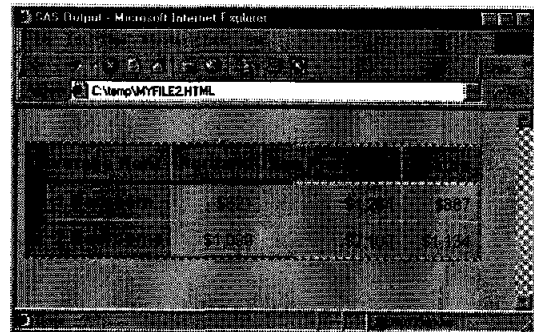


Average Rent	Seattle	Portland	San Francisco
1 Bedroom	\$887	\$621	\$1,284
2 Bedrooms	\$1,134	\$1,099	\$2,100

In version 8, the syntax is even easier:

```
ODS HTML BODY='C:\TEMP\MYFILE2.HTML';
PROC TABULATE DATA=TEMP NOSEPS
  ORDER=FORMATTED F=DOLLAR9.;
  CLASS CITY BEDROOMS;
  VAR RENT;
  TABLE RENT=' '*BEDROOMS=' ',
    CITY=' '*MEAN=' '
    / ROW=FLOAT RTS=16
    BOX='Average Rent';
RUN;
ODS HTML CLOSE;
```

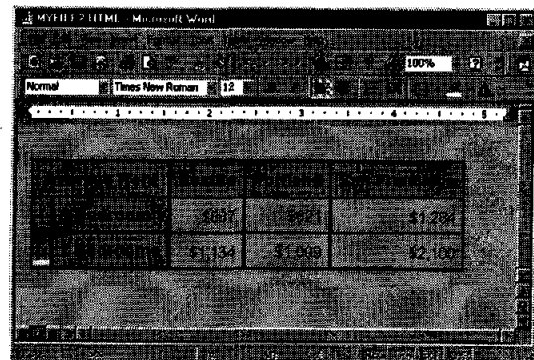
The output for version 8 is shown below:



Average Rent	Seattle	Portland	San Francisco
1 Bedroom	\$887	\$621	\$1,284
2 Bedrooms	\$1,134	\$1,099	\$2,100

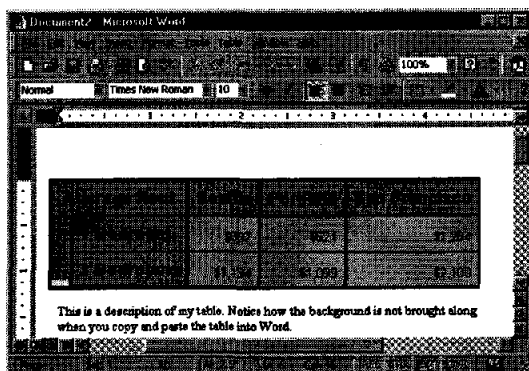
➤ TIP #18: IMPORTING INTO OTHER APPLICATIONS

Once you've created HTML output, it's easy to move your table into other applications. To move your table into a word processor, you can simply open the HTML file directly from the word processor.

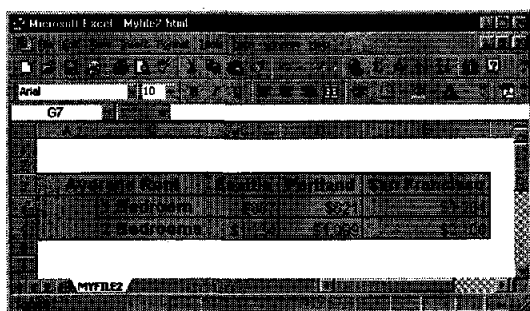


Average Rent	Seattle	Portland	San Francisco
1 Bedroom	\$887	\$621	\$1,284
2 Bedrooms	\$1,134	\$1,099	\$2,100

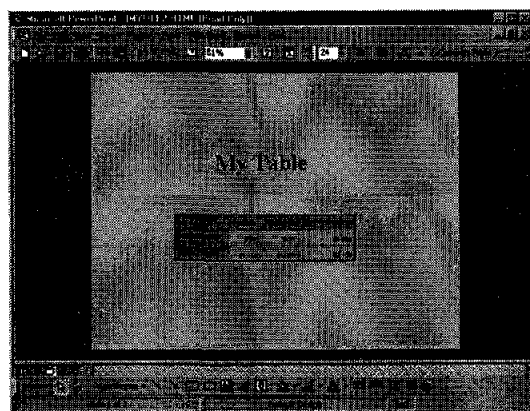
If you want to add your table to an existing document, simply open it in a browser, highlight the table, and use copy and paste to drop it into your word processor.



The same technique works well for moving your table into a spreadsheet. Just open the file directly from your spreadsheet.



You can also open an HTML file from PowerPoint. You may find that your file needs a bit of formatting once it is imported. Generally, you'll need to move the table to where you want it on the page, and you may need to resize the table and fonts.



Save this file as a PowerPoint presentation. Then you can either add the rest of your slides to this file, or import this file into another presentation.

➤ CONCLUSIONS

This paper has shown a number of tips and tricks for taking your TABULATE output to the next level. This procedure can be a powerful table generator if you take a little time to learn how to enhance your output.

There are many more ways to enhance your output. Check the SUGI proceedings for Tutorials and Coders Corner papers on the procedure. There are also more techniques in my book "PROC TABULATE by Example". Finally, you can probably figure out some new tricks on your own, based on the needs of your organization.

➤ ACKNOWLEDGEMENTS

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

Other brand and product names are registered trademarks or trademarks of their respective companies.

➤ CONTACTING THE AUTHOR

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