USING PROC MEANS

The routine PROC MEANS can be used to obtain limited summaries for numerical variables (e.g., the mean, standard deviation, etc.).

Consider the CourseEnrollments.txt data set.

```sas
DATA CourseEnrollments;
  INFILE 'T:\20165001682\ReadOnly\Data\CourseEnrollments.txt';
  INPUT Name $18. +1 Subject $4. CourseNum StartDate MMDDYY10. Section Credits Enrollment;
RUN;

PROC PRINT DATA=CourseEnrollments;
RUN;
```

A portion of the output is shown below:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Name</th>
<th>Subject</th>
<th>CourseNum</th>
<th>StartDate</th>
<th>Section</th>
<th>Credits</th>
<th>Enrollment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Christopher Malone</td>
<td>STAT</td>
<td>110</td>
<td>18141</td>
<td>1</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>2</td>
<td>Christopher Malone</td>
<td>STAT</td>
<td>110</td>
<td>18141</td>
<td>2</td>
<td>3</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Christopher Malone</td>
<td>STAT</td>
<td>210</td>
<td>18141</td>
<td>1</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>Christopher Malone</td>
<td>STAT</td>
<td>370</td>
<td>18141</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Brant Deppa</td>
<td>STAT</td>
<td>110</td>
<td>18141</td>
<td>1</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>Brant Deppa</td>
<td>STAT</td>
<td>450</td>
<td>18141</td>
<td>1</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Tisha Hooks</td>
<td>STAT</td>
<td>110</td>
<td>18141</td>
<td>1</td>
<td>3</td>
<td>39</td>
</tr>
</tbody>
</table>

The following code can be used to obtain basic summary statistics for the numeric variables Credits and Enrollment.

```sas
PROC MEANS DATA=CourseEnrollments;
VAR Credits Enrollment;
RUN;
```

The following output is produced by PROC MEANS by default:

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credits</td>
<td>31</td>
<td>3.325806</td>
<td>0.4751910</td>
<td>3.0000000</td>
<td>4.0000000</td>
</tr>
<tr>
<td>Enrollment</td>
<td>31</td>
<td>31.5483871</td>
<td>12.1979198</td>
<td>4.0000000</td>
<td>46.0000000</td>
</tr>
</tbody>
</table>
Similar to other routines, you can use a BY statement within PROC MEANS. Recall that the data set will first need to be sorted by the variable you are using in the BY statement.

```
PROC FORMAT;
   VALUE Semester 18141 = 'Fall 2009'
                    18277 = 'Spring 2010';
RUN;

PROC SORT DATA=CourseEnrollments;
   BY StartDate;
RUN;

PROC MEANS DATA=CourseEnrollments;
   VAR Credits Enrollment;
   BY StartDate;
   FORMAT StartDate Semester.;
RUN;
```

SAS also has the ability to save the output from PROC MEANS into a secondary dataset. This can be done using the OUTPUT statement, as is shown below. The new data set will be named CEMeans, and this data set will contain two new variables: AvgCredits and AvgEnrollment.

```
PROC MEANS DATA=CourseEnrollments;
   VAR Credits Enrollment;
   OUTPUT OUT=CEMeans MEAN(Credits Enrollment) = AvgCredits AvgEnrollment;
RUN;
```
The following PROC PRINT statement shows the contents of the CEMeans dataset.

```sas
PROC PRINT DATA=CEMeans;
RUN;
```

![Table showing contents of CEMeans dataset]

Note that you can prevent SAS from printing the output to the Results Viewer window by using the NOPRINT option in the PROC MEANS statement.

```sas
PROC MEANS DATA=CourseEnrollments NOPRINT;
VAR Credits Enrollment;
OUTPUT OUT=CEMeans MEAN(Credits Enrollment) = AvgCredits AvgEnrollment;
RUN;
```

At times, SAS will automatically create variables and place them in data sets for you. For example, suppose we want to obtain the average credits and average enrollment by `StartDate`. When SAS creates the CEMeans dataset, it automatically puts a variable in this dataset to identify the StartDate.

```sas
PROC MEANS DATA=CourseEnrollments NOPRINT;
VAR Credits Enrollment;
BY StartDate;
OUTPUT OUT=CEMeans MEAN(Credits Enrollment) = AvgCredits AvgEnrollment;
RUN;

PROC PRINT DATA=CEMeans;
RUN;
```

Note: SAS automatically creates a variable in the CEMeans dataset called `StartDate` so that each average can be easily identified.

![Table showing StartDate and averages]
Using our previously specified formatting for StartDate (from Handout 8), we can print StartDate, the average credits, and the average enrollments in a simple, easy-to-read data set.

```proc format;
value Semester 18141 = 'Fall'
18277 = 'Spring';
run;
```

```proc print data=CEMeans;
var StartDate AvgCredits AvgEnrollment;
format StartDate Semester.;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>StartDate</th>
<th>AvgCredits</th>
<th>AvgEnrollment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fall</td>
<td>3.31250</td>
<td>32.5000</td>
</tr>
<tr>
<td>2</td>
<td>Spring</td>
<td>3.33333</td>
<td>30.5333</td>
</tr>
</tbody>
</table>

Finally, note that there are several additional summaries you can obtain using PROC MEANS.

<table>
<thead>
<tr>
<th>Descriptive Statistic Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLM</td>
</tr>
<tr>
<td>MEAN</td>
</tr>
<tr>
<td>KURTOSIS</td>
</tr>
<tr>
<td>SUM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantile Statistic Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDIAN</td>
</tr>
<tr>
<td>P1</td>
</tr>
<tr>
<td>Q3</td>
</tr>
</tbody>
</table>

For example, the following programming statements would ask PROC MEANS to compute only the mean, median, standard deviation and range of the variables listed in the VAR statement.

```proc means data=CourseEnrollments mean median stddev range;
var Credits Enrollment;
run;
```
USING PROC FREQ

The PROC FREQ routine is analogous to the PROC MEANS routine, but it is used to obtain summaries for categorical data instead of numerical data.

For this example, consider the CarAccidents.csv data set (a portion of which is shown below).

<table>
<thead>
<tr>
<th>A</th>
<th>Gender</th>
<th>Seat Belt</th>
<th>Age Group</th>
<th>Cell Phone Involved</th>
<th>Cell Phone Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Female</td>
<td>Yes</td>
<td>16-18</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Female</td>
<td>Yes</td>
<td>Over 30</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>Male</td>
<td>No</td>
<td>16-18</td>
<td>Yes</td>
<td>Text</td>
</tr>
</tbody>
</table>

Use the Import Wizard to store this dataset in your personal library if you have not already done so. Then, print the data with a PROC PRINT statement similar to the following.

```
PROC PRINT DATA=Hooks.Car_Accidents;
RUN;
```

A partial listing of the CarAccidents data set is shown below.

<table>
<thead>
<tr>
<th>Obs</th>
<th>ID</th>
<th>Gender</th>
<th>Seat Belt</th>
<th>Age_Group</th>
<th>Cell_Phone_Involved</th>
<th>Cell_Phone_Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Female</td>
<td>Yes</td>
<td>16-18</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Female</td>
<td>Yes</td>
<td>Over 30</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Male</td>
<td>No</td>
<td>16-18</td>
<td>Yes</td>
<td>Text</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Female</td>
<td>Yes</td>
<td>18-30</td>
<td>Yes</td>
<td>Talk</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Female</td>
<td>Yes</td>
<td>18-30</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Male</td>
<td>No</td>
<td>18-30</td>
<td>Yes</td>
<td>Text</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>Male</td>
<td>Yes</td>
<td>16-18</td>
<td>Yes</td>
<td>Talk</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Male</td>
<td>Yes</td>
<td>16-18</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>Male</td>
<td>No</td>
<td>18-30</td>
<td>Yes</td>
<td>Text</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>Female</td>
<td>Yes</td>
<td>Over 30</td>
<td>No</td>
<td>None</td>
</tr>
</tbody>
</table>

Note: SAS did not like the spaces in the variable names and automatically placed a “_” between words in the column titles. Therefore, when you refer to these variables in your program, you must include the underscore (e.g., Seat_Belt is the name of the Seat Belt variable).
Consider the following summary for Gender (which is a categorical variable) obtained using the PROC FREQ routine. The TABLE command is used to specify the variables to be summarized.

```sas
PROC FREQ DATA=Hooks.CarAccidents;
   TABLE Gender;
RUN;
```

The SAS output from the PROC FREQ procedure is shown below.

Two variables can be summarized simultaneously by using a "*" between the variables to create a contingency table (i.e., a cross-tab table).

```sas
RUN;
```
There exist many options that can be specified within PROC FREQ. For example, the following program tells SAS to include only the counts in the report; that is, it removes the overall percent, row percent, and column percent from each cell of the table.

```
PROC FREQ DATA=Hooks.CarAccidents;
  TABLE Gender*Cell_Phone_Involved / NOPERCENT NOROW NOCOL;
RUN;
```

Including LIST in the options causes SAS to produce a list of counts instead of a table.

```
PROC FREQ DATA=Hooks.CarAccidents;
  TABLE Gender*Cell_Phone_Involved / NOPERCENT NOROW NOCOL LIST;
RUN;
```

Many other options exist in PROC FREQ (e.g., you can use PROC FREQ to carry out chi-square tests and several other exact tests). You can read more about these in the SAS help documentation.
USING PROC TABULATE

The PROC TABULATE routine is a popular substitute for PROC MEANS, PROC FREQ, and even PROC PRINT because the output produced can be manipulated to have a better appearance. PROC TABULATE is a sophisticated routine, and entire manuals have been written on this procedure. It is based in part on the Table Producing Language, a complex language developed by the U.S. Department of Labor.

Consider a side-by-side comparison of the PROC TABULATE code/output and the PROC FREQ code/output.

```
PROC FREQ DATA=Hooks.CarAccidents;
   TABLE Gender;
RUN;

PROC TABULATE DATA=Hooks.CarAccidents;
   CLASS Gender;
   TABLE Gender;
RUN;
```

Note one major difference in the code between PROC FREQ and PROC TABULATE: the PROC TABULATE routine requires a CLASS statement to specify any categorical variables you’d like to summarize. Note that if you are working with numerical variables, you must use the VAR statement in PROC TABULATE to specify the numerical variables you’d like to summarize.

Consider the following programming statements, which ask SAS to display both counts and percentages. The N*() and PCTN*() options are specified in the TABLE statement.

```
PROC TABULATE DATA=Hooks.CarAccidents;
   CLASS Gender;
   TABLE N*(Gender) PCTN*(Gender);
RUN;
```
The following code creates one big table that shows the counts and percentages for both Gender and Cell_Phone_Involved. Notice the counts are computed individually (i.e., a contingency table is not created here).

```sas
PROC TABULATE DATA=Hooks.CarAccidents;
  CLASS Gender Cell_Phone_Involved;
  TABLE N*(Gender Cell_Phone_Involved) PCTN*(Gender Cell_Phone_Involved);
RUN;
```

To obtain a contingency table, you should include a "*" between the variable names. Also, using two table statements will create a table of counts and a separate table of percentages in each category.

```sas
PROC TABULATE DATA=Hooks.CarAccidents;
  CLASS Gender Cell_Phone_Involved;
  TABLE N*(Gender*Cell_Phone_Involved) PCTN*(Gender*Cell_Phone_Involved);
RUN;
```
In all of the above examples involving PROC TABULATE, the variables specified in the TABLE statement appeared in columns of the resulting table. This procedure also allows for more dimensions in your table. Consider the following program and output.

```
PROC TABULATE DATA=Hooks.CarAccidents;
   CLASS Gender Cell_Phone_Involved;
   TABLE Gender, Cell_Phone_Involved;
RUN;
```

<table>
<thead>
<tr>
<th>PctN</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Female</td>
</tr>
<tr>
<td>Cell_Phone_Involved</td>
<td>No</td>
</tr>
<tr>
<td>15.21</td>
<td>32.47</td>
</tr>
</tbody>
</table>

Note that the variable that appears before the comma in the TABLE statement is placed in rows, while the variable that appears after the comma is placed in columns.

You could also request that SAS display the percentages in each group instead of the counts.

```
PROC TABULATE DATA=Hooks.CarAccidents;
   CLASS Gender Cell_Phone_Involved;
   TABLE PCTN*Gender, Cell_Phone_Involved;
RUN;
```
Note that the percentages reported so far are percentages of the total number of observations that fall in each cell. Instead, we may be more interested in calculating row percentages, as shown below.

```sql
PROC TABULATE DATA=Hooks.CarAccidents;
  CLASS Gender Cell_Phone_Involved;
  TABLE ROWPCTN*Gender, Cell_Phone_Involved;
RUN;
```

Finally, note that if you wanted to change the header for the Cell Phone Involved variable and also remove the RowPctN label, you could modify your code as follows:

```sql
PROC TABULATE DATA=Hooks.CarAccidents;
  CLASS Gender Cell_Phone_Involved;
  TABLE ROWPCTN='*Gender, Cell_Phone_Involved = 'Cell Phone Involved';
RUN;
```
To remove the empty box left behind after removing the RowPctN label, you could use the ROW=FLOAT option as follows:

```plaintext
PROC TABULATE DATA=Hooks.CarAccidents;
   CLASS Gender Cell_Phone_Involved;
   TABLE ROWPCTN=' '*Gender, Cell_Phone_Involved = 'Cell Phone Involved' /
      ROW=FLOAT;
RUN;
```

<table>
<thead>
<tr>
<th>Course Enrollments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Phone Involved</td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>Female</td>
</tr>
<tr>
<td>Male</td>
</tr>
</tbody>
</table>