



# SAS Workshop

INTRODUCTORY ASPECTS

SPRING 2011

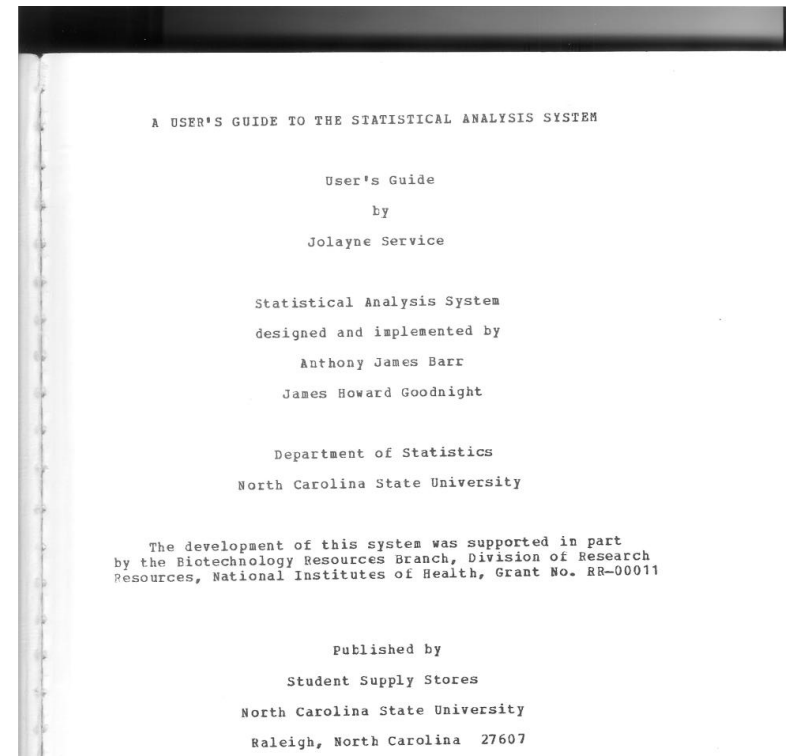
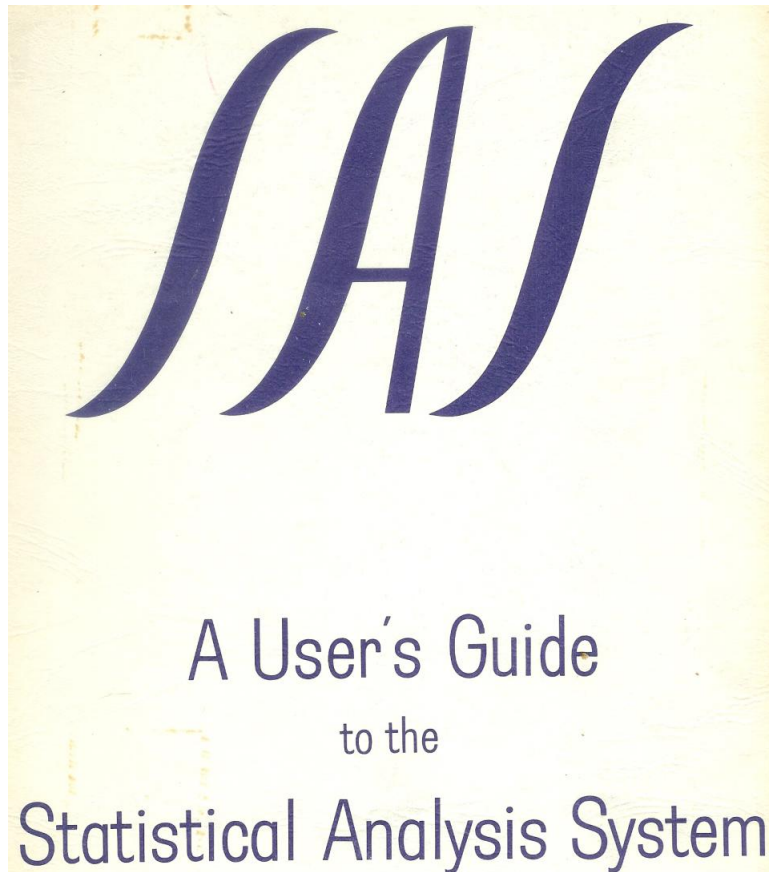
# What is SAS

- The term SAS stands for the Statistical Analysis System.
- **SAS** is a programming language as well as a set of procedures for data management and analysis.

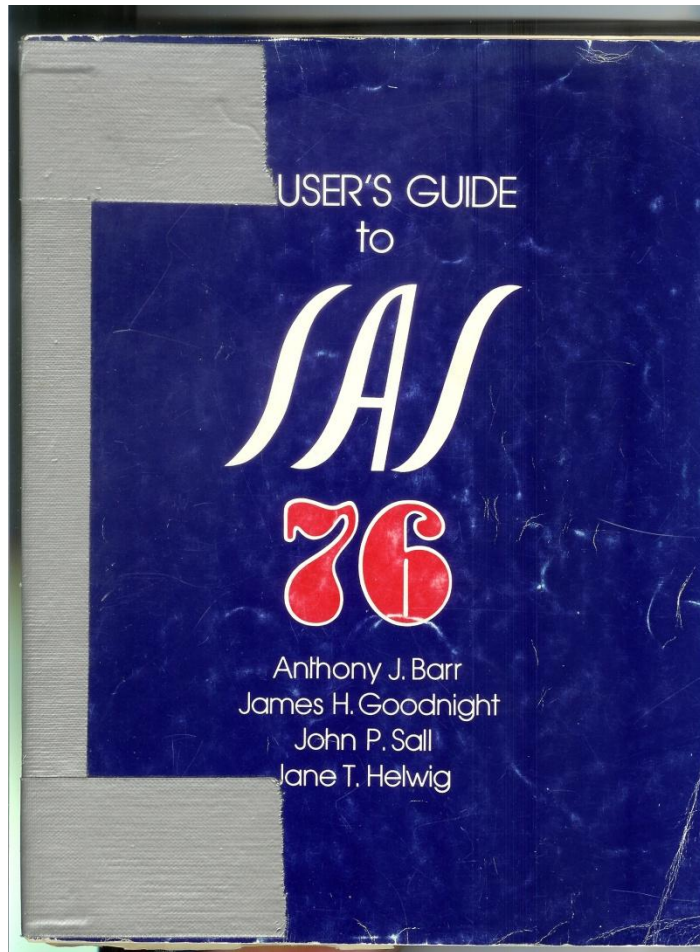
# History of SAS

- SAS was initiated on the campus of NC State circa 1966 funded by a grant. The project then moved off that campus as a private enterprise, and became SAS Institute.
- It is now one of the largest companies in the world, with sites all over the world, but the home campus is Cary, NC.

# First SAS Manual August, 1972



# SAS76 Manual



In the ten years since SAS was conceived, hundreds of people have contributed to its development through their suggestions, their support, and their use of SAS.

The faculty of the Department of Statistics, North Carolina State University, played a major role in the development of SAS. Dr. David D. Mason and Francis J. Verlinden, through their insight into the needs of statistical computing, provided the impetus for the creation of SAS. Dr. Mason's continuing support and encouragement have been vital to our progress.

Dr. Thomas M. Gerig played a key role in the multivariate aspects of SAS. Dr. Charles H. Proctor helped extensively in the development of the Guttman scaling procedure. Dr. Robert J. Monroe, Dr. Henry L. Lucas, Dr. Francis G. Giesbrecht, Dr. A. Ronald Gallant, Dr. Ardell C. Linnerud, Dr. Harvey J. Gold, and Mrs. Evelyn B. Wilson made many suggestions that found their way into the system.

The University Statisticians of the Southern Experiment Stations supported SAS and made numerous contributions over the years. We are especially indebted to Dr. E. J. Freund and Dr. Charles Gates of Texas A & M University, Dr. Robert D. Morrison of Oklahoma State University, Dr. Wilbert P. Byrd of Clemson University, Dr. Richard M. Patterson of Auburn University, Dr. William L. Sanders and Dr. T. J. Whitley of the University of Tennessee, Dr. Ramon C. Littell of the University of Florida, Dr. Clyde Y. Kramer of Virginia Polytechnic Institute and State University, Dr. Kenneth L. Koonce of Louisiana State University, Dr. D. Dal Kratzer of the University of Kentucky, and Dr. Glenn O. Ware of the University of Georgia.

Special thanks go to Dr. Shayle R. Searle, of Cornell University, and Dr. Walter R. Harvey, of Ohio State University, for their contributions to the General Linear Models procedure.

Key contributors to SAS at other installations include Dr. J. Philip Miller of Washington University; Dr. Robert Teichman of ICI United States; Dr. William J. Kennedy of Iowa State University; Dr. Ronald W. Helms of the

University of North Carolina; Dr. Daniel Chilko of West Virginia University; Dr. Gerald Hajian of American Cyanamid Inc.; Richard E. Cooper, Dr. Harold Huddleston, and William H. Wigton of the United States Department of Agriculture; Edward W. Whitehorn and Dr. Hans Schreuder of the United States Forestry Service; and Dr. David Hurst of the University of Alabama in Birmingham.

Other users who have made valuable suggestions are William Gjertsen, Frank Harrell, Kenneth Hardy, and William Reynolds of the University of North Carolina; Michael Foxworth of the University of South Carolina; Dr. Robert P. Parks of Washington University; and Dr. George C. Chao of Abbott Laboratories.

The analysis of computer performance information has grown to be a major SAS application. H. W. Barry Merrill of State Farm Mutual Insurance Company first performed large-scale SMP analyses; through his efforts, this use of SAS is now routine. Others who have made major contributions in this application area are J. Frank Chambers of Avco Financial Services, James Guthrie of the Cleveland Trust Company, and James Trefren of Abbott Laboratories.

We would like to express our appreciation to Julian Horvich of Abbott Laboratories, who founded the International SAS Users' Group and chaired its first meeting in Orlando in January of 1976.

As SAS consultants at North Carolina State University, Herbert J. Kirk and Sandra B. Donaghy have provided expertise to SAS users and such useful information to us. We are most appreciative of their help and advice.

The work of Jolayne Service, who in 1972 wrote the first comprehensive SAS user's guide, was invaluable. The form and structure of that book are found in this one also. We are deeply grateful for her contributions.

Thanks go also to Ray Danner of the National Institutes of Health, Virginia Patterson of the University of Tennessee, Richard Blocher of Washington University, Suzanne Gordon and Clifford Jennings of

# Structure of SAS Code

- **Every command in a SAS program ends in a semi-colon;** The code can be quite flexibly written. See below regarding structure of lines, use of capital letters etc.
- 
- Data One;
- Input X y Z;
- Datalines;
- 'some input data'
- ;
- Proc Print;
- Run;
- 

---

- Data one; Input X y Z; Datalines;
- 'some input data'
- ;
- proc Print; Run;

# Additional Slide

- Note, I updated as many of the double quotes as I saw. There may be others lurking so be careful. Also, when copying and pasting from slides into the editor there may be font issues that 'mess up' the double quotes.
- I added some additional data step information under some Additional Slides that demo the drop, keep and output statements.

# SAS Windows

- **Windows in SAS:** When you launch SAS it creates three windows:
- The **Program Editor** window
- The **Log** window
- The **Output** window

# SAS Windows

- **The program editor** is the window where programs are created, or read in as existing SAS code.
- **The log window** contains information about the execution of a program. It will note errors, and report other processing information regarding the execution of the program. It is very useful for debugging your program, and is a historic document of the execution that may warrant saving.
- **The output window** is where output from SAS procedures appears. The information in all of these windows can be saved to various file types.

# Variable Names in SAS

- **Variable names:** when naming variables in SAS you need to start the name with an alphabetic character A-Z. You cannot start a name with a number or other character, but numbers and some characters are allowed. However, you cannot use arithmetic operators in a variable name.
- Typically names are best limited to say 10 or fewer characters, and if possible those characters should relay some aspect of what the variable represents.
- Some Valid names: Sex, Gender, Pre\_I, BIG, big, Big, V1, V2, V3, etc.

# Data types

- One way to view variables is whether they are numeric valued or character 'valued'.
- SAS requires that you identify character variables: the default is numeric.
- For example, Gender is a categorical variable, essentially nominal. However, it could be coded with values as (M, F), or (Male, Female). Another coding might be to use (0 =male, 1=female). There are benefits and problems with each coding approach.

# Other Data Types

- SAS also has date and time formats for variables but we will not discuss those in this presentation.
- There are statements in SAS that can establish the length and type of a variable. Again we will not discuss that in the presentation.

# SAS Processing

- SAS views working with data as a dichotomy consisting of a Data Step and then a Procedure (Proc) step.
- In the Data Step the variables and data are defined and input into a SAS file.
- The Proc steps are actions upon that SAS data file.

# Data Files

- Every now and then one hears the term flat file. That term relates to the two dimensional nature of many data files.
- The most common structure is for each row in a file to correspond to a individual's record, and the columns in that file correspond to variables.

# Input-Output Processes

- When data are brought into SAS they are brought in during the data step. This data step is a sequential process where each data record is read in one at a time processed (assignment statements, logical statements and arithmetic statements are executed) and then output to the SAS file being created.
- Once an input record is processed and output, a new record is read and processed. This sequence continues until all the data in the input file are read in and output.

# Data Sheet

Student_ID	Gender	Test1	Test2	F_Exam
1	M	75	78	76
2	M	90	85	87
3	M	85	76	72
4	M	86	88	90
5	M	92	95	93
6	M	87	74	77
7	M	65	76	72
8	M	76	78	74
9	M	90	92	96
10	M	100	96	88

# Assignment Statements

- Assignment statements are SAS statements that assign a value to a variable.
- $X=5;$
- $Y=35/10 + 3;$
- `Tex="Hello";` \*character assignment requires double quotes;

# Arithmetic Operations

- **Order of Arithmetic in assignment Statements:** exponentiation has the highest order followed by multiplication or division, then by addition or subtraction.
- The use of parentheses is to override a particular order— statements in parentheses are evaluated first.

# Arithmetic Operations

- Consider the following variables for example:  $X=5$ ;
- $W=X**2$ ; yields  $W=25$
- $V=X**2/2$ ; yields  $V=12.5$
- $P=5+X**2/2$ ; yields  $P=17.5$
- $Q=(5+X)**2/2$ ; yields  $Q=50$
- $R=(5+X)**(2/2)$ ; yields  $R=10$

# Annotation of Code

- It always pays to add some notes in the code regarding what you are doing.
- Lines with annotation begin with an asterisk and end with a semi-colon
- \*Example of Annotation line;

# Examples of The Data Step

- **\*Creating a data file from scratch;**
- **data** one; \*starts data step;
- **x=5;**
- **y=3;**
- **z=x/y;** \*ends data step;
- **proc print;** \*procedure steps;
- **run;**

# Examples of The Data Step

- **\*Using the output statement to create different records;**
- `data two; *starts data step;`
- `x=5;`
- `y=3;`
- `z=x/y;`
- `output;`
- `x=3; y=5; z=x/y;`
- `output; *End of data step;`
- `proc print; *Procedure step;`
- `run;`

# Examples of The Data Step

- **\*Reading the data from records in the program;**
- Data Three; \*start of data step;
- **Input ID \$ x y;**
- $Z=x/y;$
- **Cards;**\*can use **datalines** statement here as well;
- A 5 3
- B 3 5
- C 2 7
- D 9 4
- E 6 5
- ;;;; \*end of data step;
- Proc print; \*start procedure steps;
- Run;

# Examples of The Data Step

- **\*Reading data from an external file;**
- **Filename datum** “C:\Documents and Settings\obrienk\Desktop...”;
- **\*tells SAS from where to read the data;**
- **Data four;**
- **Infile datum;**
- **Input ID \$ X Y;**
- **Z=X/Y;**
- **Proc print;**
- **Run;**

# Examples of The Data Step

- **\*Reading from an external file and Writing that file to a SAS database;**
- Filename datum “D:\Data\Sas\_Workshop\trial.txt”; \*Location of external file to read into SAS;
- **Libname SASDAT** “D:\data\SAS\_Workshop” ; \*tells SAS where to put the database;
- \*Associates the name SASDAT with a folder or other location on your computer;
- Data five; infile datum;
- Input ID \$ XY;
- Z=X/Y;
- **Data SASDAT.Trial;** set five;
- \*tells SAS to create the database Trial from our work file called Five,
  - and Store it in the specified library SASDAT;
- Run;

# Examples of The Data Step

- **\*Reading data from an Existing SAS database;**
- Libname SASDAT “D:\data\SAS\_Workshop”  
;
- \*Associates the name SASDAT with a folder or other location on your computer;
- Data seven; **Set SASDAT.Trial;**
- \*creates working file Seven from the existing SAS database Trial;
- Proc print;
- Run;

# Examples of The Data Step

- **\*Streaming data lines;**
- Data eight;
- Input ID \$ XY @@; \*The double ampersands tell SAS the data are being streamed;
- Z=X/Y;
- Cards; \*again, could use the datalines statement as well;
- A 5 3 B 3 5 C 2 7 D 9 4 E 6 5
- ;;;
- Proc Print;
- Run;

# Examples of The Data Step

- Now some examples of merging files or concatenation of files.
- Consider the typical view of a data file (flat file) where the rows represent the records of items or subjects, while the columns represent variables containing information on each item.

# Examples of The Data Step

- SAS will import data from other programs like Excel, SPSS etc.
- Although you can use program statements to do this, the easiest approach is to use the Import feature in the file menu.
- Lets look at an example of this approach.

# Examples of The Data Step

- We will use the Excel file called Grades. This file has three data sheets in it.
- Sheet1 is named Males
- Sheet2 is named Females
- Sheet3 is named Grades

# Males Data Sheet

Student_ID	Gender	Test1	Test2	F_Exam
1	M	75	78	76
2	M	90	85	87
3	M	85	76	72
4	M	86	88	90
5	M	92	95	93
6	M	87	74	77
7	M	65	76	72
8	M	76	78	74
9	M	90	92	96
10	M	100	96	88

# Importing Data Example

- ```
/******  
/* import the Excel file */  
/******  
proc import datafile=  
"D:\Data\Sas_Workshop\Grades.xls"  
out=Males;  
sheet="Males";  
getnames=yes;  
run;
```

# Importing Data Example

- Or we use the import option under the file menu as follows:
- Go to File Menu and select Import Data
- Choose Excel format
- Follow along with the dialogs. Here we will want to use the Work Library (active library) and then give the file a name—  
Males
- Then just go to finish.

# Importing Data Example

- Let's print this file—Males
- Now bring in the female data
- Let's print this file—Females
- Now bring in the Homework
- Print that file as well.

# Note on Active File and Procedures

- Note that the active data file in SAS is the last one created or used in a procedure.
- So the last file we created was Homework. If I ran the statements: Proc Print; run; It would print the file Homework.
- If I wanted to print the Females file I would write: **Proc Print Data=Females; Run;**

# Merging and Concatenating Files

- Recall that the typical file structure is for the rows to represent different records (or individuals) while the columns contain the variables of interest.

# Merging Files

- A merging of files in SAS is to add new variables or to update existing variable information.

# Concatenating Files

- A concatenation in SAS is to add new records to a file.
- In a concatenation, the variables in each data file must typically be the same in regard to name and to type.

# Merging and Concatenating Files

- In most every software package to do a merge requires that the data be sorted by an ID variable that is unique in that there are no repeat values.
- In SAS the procedure for sorting is PROC SORT;
- A By statement is required which tells SAS which variable to sort the data by.

# Concatenating Files

- The process for concatenation is simply to create a new data file that contains the files you want to concatenate.
- Let's concatenate the male and female data files.
- **DATA COMBO; SET Females Males;**
- **Proc Print; run;**

# Example: Proc Sort

- We'll sort the Homework file by Student ID.
- Proc Sort Data=Homework; By Student\_ID; Run;

# Merging Files

- The statements to merge the new Homework information to the combined male and female files are:
- Data All; Merge Combo Homework;
- By Student\_ID;
- Proc Print;
- Run;

# Assignment Statements

- Lets use an assignment statement to assign the score for the class that combines the homework and the test scores.
- The syllabus read that the score was comprised of 25% for each test, 40% for the final and 10% for homework.

# Assignment Statement

- We can code the score as follows:
- $\text{Score} = 0.25 * \text{Test1} + 0.25 * \text{Test2} + 0.40 * \text{F\_Grade} + 0.10 * \text{HW\_Avg};$
- Lets enter this run the statement and then print out the file.

# Assigning Score

- **Data all;**
- **set all;**
- **\*Tells SAS I want to work on this file;**
- **Score=0.25\*Test1+0.25\*Test2+0.40\*\_Grade+0.10\*HW\_Avg;**
- **proc print;**
- **run;**

# If-Then Statements

- If-then statements are used to control how SAS works on a data file depending on conditions we set.
- These statements have the form:
- **If** this **then** that;

# If-Then Statement

- To assign a letter grade we can use the if-then statement to
- Lets suppose we want to assign a letter grade on a 10 point scale that is based on the score we just computed.

# Assign Final Letter Grade

- If  $\text{score} < 59.5$  then  $\text{L\_Grade} = \text{"F"};$
- If  $59.5 \leq \text{score} < 69.5$  then  $\text{L\_Grade} = \text{"D"};$
- If  $69.5 \leq \text{score} < 79.5$  then  $\text{L\_Grade} = \text{"C"};$
- If  $79.5 \leq \text{score} < 89.5$  then  $\text{L\_Grade} = \text{"B"};$
- If  $\text{Score} \geq 89.5$  then  $\text{L\_Grade} = \text{"A"};$

# Assignment of Letter Grade

- **Data All;Set All;**
- **\*Assign Letter Grade on 10 point scale;**
- **If score<59.5 then L\_Grade="F";**
- **If 59.5<=score<69.5 then L\_Grade="D";**
- **If 69.5<=score<79.5 then L\_Grade="C";**
- **If 79.5<=score<89.5 then L\_Grade="B";**
- **If Score>=89.5 then L\_Grade="A";**
- **proc print; title "Final Letter Grade for the Course";run;**

# Additional Slide

- Recall the `Output` statement used earlier. Let's look at using it to recreate the individual files that comprise `Data All`.

# Additional Slide

- Data males2 Females2;
- If Gender="M" then output Males2;
- If Gender="F" then output Females2;
- \* Those statements created two files;
- \*The Male file and the female file;
- Proc Print data=Males2;
- Proc Print data=Females2;
- Run;

# Additional Slide

- Notice in the print output the grade information is still in each gender specific file. If I want to drop Variables L\_Grade, HW\_Avg, Score and F\_Exam, I can use the statements on the following slide.

# Additional Slide

- Data Males2;set Males2;
- Drop L\_Grade HW\_Avg Score F\_Exam;
- Data Females2;Set Females2;
- Drop L\_Grade HW\_Avg Score F\_Exam;
- Proc print; \* Prints Female2 by default;
- Run;

# Additional Slide

- Now lets take care of the Grades file.
- Data Grades2; Set All;
- Keep Student\_ID HW\_Avg Score  
F\_Exam;
- \*retains only the original 3 variables;
- Proc Print;
- Run;

# Additional Slide

- So we see how to use the **output** statement again, and the new **Keep** and **Drop** statements in the Data Step.
- Also notice that you can specify several new working data files in a single data statement.

# Descriptive Statistics

- Let's look at three procedures that provide basic statistical descriptions. These are not graphical but rather summary indices like frequencies, means, medians, standard deviations, etc.
- The Three are: Proc Freq; Proc Means;
- and Proc Univariate;

# Proc Freq;

- Proc Freq will provide the frequency distribution for each variable in the data file.
- It is a very good way to check for odd or incorrect values before going into any further analyses.

# Proc Freq;

- The most simple syntax is to say Proc Freq;
- However, you might want to use additional syntax to limit the set of variables being examined, or to get frequency distributions for subsets of the data.

# Proc Freq;

- Let's run the Frequencies for the data file we created through concatenation of Males and Females and the merger with Grades.
- Proc Freq Data=All; Run;
- Note that every variable is represented.

# Proc Freq;

- Let's delimit the variables to just the Final Exam, HW\_Avg and the L\_Grade
- Proc Freq Data=All;
- Tables HW\_Avg F\_Exam, L\_Grade;
- \*the tables statement delimits the variables for which frequencies are provided;
- Run;

# Contingency Tables or Cross Tabulations

- Suppose we wanted to see if males and females had similar letter grade distributions.
- Since both variables are categorical we can best look at this in a contingency table format.
- Title "Crosstabulation of Letter Grade by Gender";
- **Proc Freq Data=all; Tables gender\*L\_Grade/chisq;**
- **run;**

# Proc Means

- Provides the means and simple descriptive statistics for quantitative variables (interval or ratio scale).
- There is an associated Var statement that delimits the variables you receive information about. Without the Var statement you get the means for all variables in the file.

# Proc Means

- Let us run two different proc means for the data file All;
- Proc means data=all; run;
- And then---
- Proc means data=all;
- Var test1 test2 F\_exam; run;

# Proc Univariate

- **Proc univariate** provides a wide range of descriptive statistics, confidence intervals, percentiles, and some statistical tests about the mean or center of a distribution.
- It also uses a **Var** statement to delimit the analysis to those variables of interest.

# Proc Univariate

- In addition one can use a **Class** statement that will compute the statistics for each level of a group defining variable.

# Proc Univariate

- Let's run the following:
- **proc univariate data=all;  
var test1; run;**
- **proc univariate data=all; var test1;  
class gender ;run;**

# End of this section

- Thanks for your attention and best of luck with your work with SAS.
- Remember you can always contact the Dept of Biostatistics regarding help with SAS.