

1. Introduction to SAS

A first look at

- **SAS data sets and data libraries**
- **Rules for SAS statements and SAS names**
- **Constructing and running SAS programs**

What is SAS?

- SAS originally stood for “Statistical Analysis System”
- At its core, SAS is a programming language for
 - Managing data:
 - reading
 - transforming
 - manipulating
 - combining (e.g. merging)
 - Reporting on data:
 - data lists
 - summary tables
 - graphics
 - Analyzing data:
 - data summarization
 - complex statistical modeling (not the focus of this course)
- SAS (a.k.a. The SAS System) is a collection of products from SAS Institute in Cary, NC

Module	Purpose
Base SAS	Data organization, transformation, and reporting
SAS/STAT	Advanced statistical analysis
SAS/GRAPH	Graphical tools
many more...	

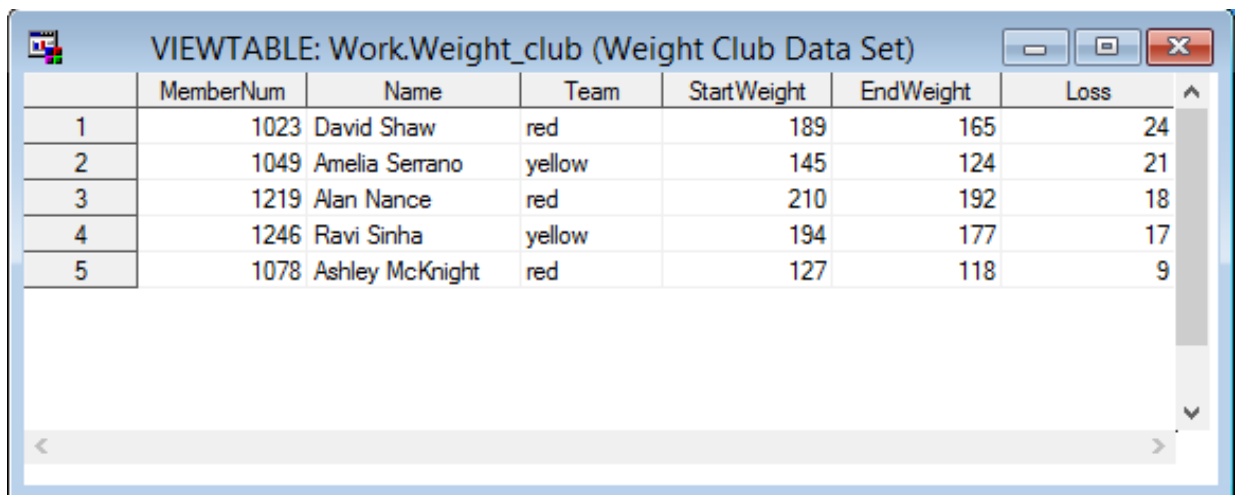
Important Types of SAS Files:

- SAS data sets -- File extension = *.sas7bdat
- SAS programs -- File extension = *.sas
- SAS log files -- File extension = *.log

SAS data sets:

- A SAS data set contains data values that are organized as a table of observations (rows) and variables (columns) that can be processed by SAS software.
- A SAS data set also contains descriptor information such as the data types and lengths of the variables.

Figure 1: View of “work.weight_club” dataset



	MemberNum	Name	Team	StartWeight	EndWeight	Loss
1	1023	David Shaw	red	189	165	24
2	1049	Amelia Serrano	yellow	145	124	21
3	1219	Alan Nance	red	210	192	18
4	1246	Ravi Sinha	yellow	194	177	17
5	1078	Ashley McKnight	red	127	118	9

Important Data Set Terminology

- Each row of a SAS dataset is called an **observation**.
- Each column of a SAS dataset is called a **variable**.

Where might a SAS data set come from?

1. Importing data from a text file (e.g. csv file)
2. Importing data from a Microsoft excel spreadsheet
3. Extracting data from an electronic database
4. Direct data entry through a SAS program
5. Running SAS code that processes existing SAS data sets and produces new ones

Figure 2: SAS Program for Data Entry using DATALINES statement

```
data weight_club(label='Weight Club Data Set'); (1)
  input MemberNum 1-4 Name $ 6-24 Team $ StartWeight EndWeight; (2)
      Loss = StartWeight - EndWeight; (3)
datalines; (4)
1023 David Shaw          red      189    165
1049 Amelia Serrano      yellow  145    124
1219 Alan Nance          red      210    192
1246 Ravi Sinha          yellow  194    177
1078 Ashley McKnight     red      127    118
; (6)
run; (7)
```

- (1) The DATA statement tells SAS to begin building a SAS data set named weight_club.
- (2) The INPUT statement identifies the fields to be read from the input data and names the SAS variables to be created from them.
- (3) This is an assignment statement that calculates the weight each person lost and assigns it to a new variable, Loss.
- (4) The DATALINES statement indicates that lines of data follow.
- (5) These data lines contain the raw data. This way of reading raw data is useful when you don't have a lot of data.
- (6) The semicolon signals the end of the raw data.
- (7) The RUN; statement asks SAS to go ahead and execute the group of statements beginning with the word DATA.

SAS Programs

- SAS programs are made up of **statements** that must follow certain rules in order for SAS to understand them.
- Rules for SAS Statements
 - Most SAS statements begin with an identifying keyword.
 - All SAS statements end with a semicolon.
 - SAS statements are not case sensitive.
 - SAS statements are free format.
 - They can begin anywhere on a line and end anywhere on a line.
 - One statement can continue over several lines as long as you do not split a word over two lines.
 - Several statements can be on one line.
 - Words in SAS statements are separated by blanks – as many as you want – or by special characters.
 - An example of a special character is the = sign in the creation of the “Loss” variable above.
- Rules for Most SAS Names (data sets, variables)
 - A SAS name can contain from 1-32 characters.
 - The first character must be a letter or underscore.
 - Subsequent characters can be letters, numbers, or underscores.
 - Blanks cannot appear in a SAS name.
 - SAS names are NOT case sensitive.

SAS Programming Style Recommendations:

- Programming Style (applicable in most cases):
 - Start each SAS statement on a new line.
 - Start DATA and PROC statements in column 1 (e.g., left of page).
 - Indent other statements within the DATA or PROC step to indicate the logical structure of the step.

Figure 3: Example Acceptable and Unacceptable SAS Code

Sample Code	How Ye Be Judged
<pre> data percent_change; set weight_club; format percChange 7.2; percChange = Loss / StartWeight * 100; if percChange > 10.0 then success = 'Y'; else success = 'N'; run; </pre>	Acceptable
<pre> DATA percent_change; SET weight_club; FORMAT percChange 7.2; percChange = Loss / StartWeight * 100; IF percChange > 10.0 THEN success = 'Y'; ELSE success = 'N'; RUN; </pre>	Also fine
<pre> data percent_change; set weight_club; format percChange 7.2; percChange = Loss / StartWeight * 100; if percChange > 10.0 then success = 'Y';else success = 'N';run; </pre>	Not Acceptable (and very easy to grade)

- Variable Naming Style:
 - Permanent variable names should convey meaning.
 - Temporary variable names should be short.
 - Variable names should not be longer than necessary.
 - Example variable names (Starting Weight):
 - StartWeight (acceptable)
 - startWeight (the same name)
 - start_weight (acceptable)
 - SW (bad)
 - SAS will display/print a variable name according to how it's first defined

Figure 4: Sample Code (Manually creating “Loss Category” Dataset)

```

/* Example with permanent and temporary variables */
/* note that temporary means we do not want to keep the variable, not
   that the variable is automatically deleted for us */
data lossCategory(label="Loss Categories");

  do i = 1 to 5 by 1; /* the i variable is a temporary loop variable */
    lowerLimit = 5*(i-1);
    upperLimit = 5*i-1;
    output;
  end;

  drop i; /* drop the i loop variable */
run;

```

	lowerLimit	upperLimit
1	0	4
2	5	9
3	10	14
4	15	19
5	20	24

DATA Steps and PROC Steps

- Within a SAS program, most statements occur in the context of either a DATA step or a PROC step.
 - A DATA step is code that typically creates a data set
 - A PROC step is code that typically produces a report or performs an analysis
- Statements that do not are called *global* statements, and we will talk about those later.
- Almost all SAS programs consist of a series of DATA steps and PROC steps.
- Each PROC step calls a SAS *procedure*
 - SAS procedures are preprogrammed routines for producing a certain type of report or doing a particular type of analysis.
 - The word after PROC names the procedure.
- Examples (we will learn about):
 - The PRINT procedure produces data listings.
 - The TABULATE procedure produces data summaries for continuous variables.
 - The FREQ procedure produces data summaries and statistical analysis for count data.

Figure 5: Example DATA and PROC steps with HTML output

```
/* example PROC PRINT step TITLE statement */  
title "PROC PRINT Output - Health Club Data";  
proc print data=weight_club;  
run;
```

```
/* example PROC TABULATE step */  
/* summarize continuous variables by team */  
title "PROC TABULATE Output";  
title2 "Mean Initial Weight, Final Weight, and Weight Loss";  
proc tabulate data=weight_club;  
  label StartWeight = 'Initial Weight'  
        EndWeight   = 'Final Weight'  
        Loss        = 'Weight Loss';  
  class team;  
  var StartWeight EndWeight Loss;  
  table team, MEAN*(StartWeight EndWeight Loss);  
run;
```

PROC PRINT Output - Health Club Data

Obs	MemberNum	Name	Team	StartWeight	EndWeight	Loss
1	1023	David Shaw	red	189	165	24
2	1049	Amelia Serrano	yellow	145	124	21
3	1219	Alan Nance	red	210	192	18
4	1246	Ravi Sinha	yellow	194	177	17
5	1078	Ashley McKnight	red	127	118	9

PROC TABULATE Output
Mean Initial Weight, Final Weight, and Weight Loss

	Mean		
	Initial Weight	Final Weight	Weight Loss
Team			
red	175.33	158.33	17.00
yellow	169.50	150.50	19.00

- Common Elements of PROC steps:
 - A PROC statement, which includes the word PROC, the name of the procedure you want to use, and the name of the SAS data set that contains the data to be analyzed.
 - Additional statements that give SAS more information about what you want to do, for example, the CLASS, VAR, TABLE, and TITLE statements.
 - *What is different about the TITLE statement?*
 - A RUN statement, which indicates that the preceding group of statements is ready to be executed.
 - Note: a DATA statement or another PROC statement would have given the same signal.
 - ALWAYS USE A RUN STATEMENT.

Figure 6: Using a PROC Step to Signal Program Execution

```
/* Malfunctioning Code Example */
title "PROC PRINT Output - Health Club Data";
proc print data=weight_club;

title "PROC TABULATE Output";
title2 "Mean Initial Weight, Final Weight, and Weight Loss";
proc tabulate data=weight_club;
  label StartWeight = 'Initial Weight'
        EndWeight   = 'Final Weight'
        Loss        = 'Weight Loss';
  class team;
  var StartWeight EndWeight Loss;
  table team, MEAN*(StartWeight EndWeight Loss);
run;
```

- So a DATA step or PROC step is composed of SAS statements that make sense in that context.
- Chapter 2 will introduce us to some basic SAS PROCs.
- Chapters 3-5 will focus on the DATA step.
 - Becoming proficient in your use of the DATA step is the most important goal of this course.
 - The programming you can do in the DATA step provides SAS's powerful data management capabilities: creating new variables, combining data sets, rearranging data sets.
 - You CANNOT become proficient in a programming language unless you repeatedly write the same code from scratch.

Combining SAS Steps into a Program

- A SAS program consists of one or more DATA and/or PROC steps which are submitted to the computer together.
- SAS steps are processed sequentially
- A program can have more than one DATA step and create more than one SAS data set.
- A SAS data set can be used in turn by more than one PROC.
- A PROC can use a SAS data set created in a previous program.

More about SAS Data Sets and Variables

- How big can a SAS data set be?
 - No limit for the number of observations
 - No limit for the number of variables
 - Data sets are stored on disk so there are practical limits
- Variable types
 - Only two types: numeric or character
 - Numeric variables:
 - Used to store values that you want to calculate with, such as age and weight.
 - SAS always stores numeric values – even integers - with *floating point representation** in from 3-8 bytes (8 is the default).
 - Character variables:
 - Use to store any strings of letters, digits, or special characters, such as name or SSN.
 - The *ASCII representation*** of each character is stored in one byte of space, and length can be from 1 to 32,767 bytes.
 - Character variables need to be as long as required (as many bytes as needed) to store the longest possible value the variable might have.

* With floating point, a value is represented as a number – the mantissa – raised to a power – the exponent. Under Windows, the mantissa is stored in 52 bits and the exponent is stored in 11 bits. One bit is left over for the sign. $52 + 11 + 1 = 64$ bits = 8 bytes. Floating point representation allows SAS both to store very large numbers and to perform calculations that require many digits of precision to the right of the decimal.

** In ASCII coding, each possible character (a, b, c, ... A, B, C, ... 0, 1, 2, ... +, -, =, @, ...) has a unique 8-bit (one byte) code. There are 256 codes (2 to the 8^{th} power) available but only about 100 characters that need to be represented.

Missing values:

- Most collections of data contain missing values. That is, for some variables for some observations, there is no value. How does SAS represent missing values?
- SAS stores a dot (.) for a missing value of a numeric variable (by default)
 - This is different from the character string “.”
 - Specifying “.” as a numeric missing value is very common programming error!!
 - There are other types of missing numeric values too: .A, .B, ..., .Z (we’ll discuss later)
- Missing numeric values are not 0. They are excluded from arithmetic and statistical calculations.
- SAS always stores a blank (“ ”) for a missing value of a character variable.
- Each SAS procedure checks variables for missing values and handles them in some predefined way.
 - See PROC documentation for more details.
 - For example, PROC REG will excluding observations from linear regression analysis when one of the variables is missing.

Data and Descriptors

- SAS data sets contain more than data. They also contain “descriptors” (i.e., metadata) about the data set as a whole and about the variables contained in the data set.
- The “data” portion of a SAS data set can be visualized using PROC PRINT (for example)
- The descriptor portion of a dataset can be visualized using PROC CONTENTS

Figure 7: Basic Use of PROC CONTENTS with Partial HTML Output

/* example PROC CONTENTS step */ proc contents data=weight_club; run;			
Data Set Name	WORK.WEIGHT_CLUB	Observations	5
Member Type	DATA	Variables	6
Engine	V9	Indexes	0
Created	08/17/2016 14:58:31	Observation Length	64
Last Modified	08/17/2016 14:58:31	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	YES
Label	Weight Club Data Set		
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
5	EndWeight	Num	8
6	Loss	Num	8
1	MemberNum	Num	8
2	Name	Char	19
4	StartWeight	Num	8
3	Team	Char	8

Sort Information	
Sortedby	MemberNum
Validated	YES
Character Set	ANSI

SAS Data Libraries

- Every SAS data set is stored in a SAS data library.
 - A SAS data library is a collection of SAS data sets recognized as a unit by the SAS System.
 - In a directory-based operating system such as Windows or Unix, a SAS data library is a collection of SAS data sets of the same engine type, stored in a specific directory (or folder).
 - Other types of files that are not datasets can reside in the same folder.
 - Any number of SAS data sets can be stored in a single library.
 - Any number of SAS data sets from any number of libraries can be used, created, and deleted in a single program.
- In SAS programs, data sets are referenced with a two-level name.
 - The first level identifies the SAS library.
 - The second level identifies the member (i.e. the data set).
 - The syntax for referring to a dataset is ***libref.filename***, where
 - *libref* is the name of the SAS library, and
 - *filename* is the SAS data set name.
- Every SAS session has a default library called the WORK library.
 - To create or utilize data sets in this library, one can omit the *libref* component of the two-level name.
 - All WORK library members will be deleted at the end of the SAS session.

The LIBNAME Statement

- SAS libraries are created using the LIBNAME statement.
- Once a library has been created, it can be used until it is cleared.
- General form of the LIBNAME statement:

LIBNAME *libref* <engine-name> "SAS-data-library";

- *libref* -- any valid SAS name (but only up to 8 characters long)
- engine-name -- optional parameter specifying one of the library engines supported by a given operating system
 - V9 -- (default) accesses Version 8 or 9 SAS data sets (extension .sas7bdat)
 - XPORT -- accesses transport format files
 - We will always use V9 (and so this can be omitted)

Figure 8: Using a LIBNAME statement with Partial SAS Log

```
/** example code to create a sas library and read a dataset */
libname bios511 "C:\Users\psioda\Documents\Teaching\BIOS-511-FALL-2016\course-data";

/** also fine (potentially) */
/** '.' references the current working directory in the SAS session */
libname bios511 v9 ".\course-data";

/** prevents overwriting */
libname bios511 ".\course-data" access=read;

proc print data=bios511.candy;
run;

/** clears the SAS library */
libname bios511 clear;
```

Log - (Untitled)

```
1 libname bios511
1 ! "C:\Users\psioda\Documents\Teaching\BIOS-511-FALL-2016\course-data";
NOTE: Libref BIOS511 was successfully assigned as follows:
      Engine:          V9
      Physical Name:
      C:\Users\psioda\Documents\Teaching\BIOS-511-FALL-2016\course-data
```

Temporary versus Permanent SAS Data Sets

- You can store or reference SAS data sets in the default WORK library by omitting the libref or by using the *libref* WORK .

<code>data one;</code> <code>infile xyz;</code> <code>input a b c;</code> <code>run;</code>	=	<code>data work.one;</code> <code>infile xyz;</code> <code>input a b c;</code> <code>run;</code>
<code>proc print data=one;</code>	=	<code>proc print data=work.one;</code>

- You can permanently store SAS data sets by using a *libref* other than WORK. The folder where you want to store your data sets must exist.
- To work with a permanent data set after you or someone else has created it, you must first establish a *libref* to the folder containing the data and then use a two-level name.

```
libname funHW "c:\BestClassEver";

data funHW.one;
    infile xyz;
    input a b c;
run;

proc print data = funHW.one; run;
```

- The temporary WORK library exists only for the duration of your SAS session and is cleared out between sessions.
- The availability of the WORK library is a GOOD THING, as most data sets you will ever create are of only temporary value and do not need to be saved permanently.

Comments in SAS Code

- There are two ways to insert comments in SAS programs:
 1. `* message ;`
 - a. Must be written as a separate statement and cannot contain internal semicolons.
 2. `/* message */`
 - a. Can be written within statements or anywhere a blank can appear.
 - b. These comments can contain internal semicolons.
- Comments can be used anywhere in a SAS program for documentation purposes.
- SAS ignores comments during processing.

Figure 9: How to Comment SAS Code

```
libname funHW v9 "c:\BestClassEver"; ** create a permanent
library;

/** create the funHW.one dataset **/
data funHW.one;
    infile xyz;
    input a b /* comment */ c;
run;

proc print data=one; * comment; run;
```

- Do not comment trivial code.
- Do not re-write the SAS documentation.
- Always comment moderately complicated code (e.g., custom algorithms you have written).
- **Always write a summary of the programs purpose at the top (i.e. a header section).**

Figure 10: Example Program Header

```
*****
*   TITLE :      Chapter # 1 Supporting Program
*
*   DESCRIPTION:  SAS code from notes for Chapter 1
*
* -----
*   JOB NAME:    chapter1.SAS
*   LANGUAGE:    SAS, VERSION 9.4
*
*   NAME:        Matthew Psioda
*   DATE:        2017-08-07
* -----
*
*   Honor Code Pledge:  On my honor, I have neither given nor received
*   unauthorized aid on this assignment.
*
*   Signature: [YOU MUST ALWAYS COMPLETE THIS SECTION]
*
*****;

FOOTNOTE "Job chapter1.SAS run by <your name> on &SYSDATE at &SYSTIME" ;
OPTIONS MERGENOBY=WARN NODATE NONUMBER LS=95 PS=54;
```

What happens when I run a SAS program?

- When you run a SAS program, the results generated by SAS are divided into two parts: SAS Log and SAS Output.
 - SAS Log
 - Contains information about the processing of the program.
 - Prints the statements you submitted, including comments.
 - Prints any error and warning messages generated during program execution.
 - Prints notes relating to each step. For example,
 - For each DATA step, documents the creation of the data set.
 - For each PROC step, indicates how much time the procedure spent operating.
 - SAS Output
 - Contains the results of the PROC steps.

Figure 11: Simple SAS Program

```
TITLE1 'Chapter 1 Example: A Simple SAS Job';

LIBNAME classlib 'P:\Teaching\BIOS-511\course-data';

DATA work.class;
  SET classlib.class;

  htmet = ht*2.54;
  wtmet = wt/2.2;

  LABEL htmet = 'Height in Centimeters'
        wtmet = 'Weight in Kilograms';
RUN;

TITLE2 'Listing of Data Portion of Data Set WORK.CLASS';
PROC PRINT DATA=work.class;
RUN;

TITLE2 'Listing of Descriptor Portion of Data Set WORK.CLASS';
PROC CONTENTS DATA=work.class;
RUN;
```

Figure 12: Simple SAS Program Log

```

733
734 *****
735 *   TITLE :       A Simple SAS Program
736 *
737 *   DESCRIPTION: A basic program using the "class" dataset
738 *
739 *-----
740 *   JOB NAME:     simple-program.SAS
741 *   LANGUAGE:     SAS, VERSION 9.4
742 *
743 *   NAME:         Matthew Psioda
744 *   DATE:         2017-08-07
745 *-----
746 *
747 *   Honor Code Pledge: On my honor, I have neither given nor received
748 *   unauthorized aid on this assignment.
749 *
750 *   Signature:
751 *
752 *****;
753
754 FOOTNOTE "Job chapter1.SAS run by <your name> on &SYSDATE at &SYSTIME" ;
755 OPTIONS MERGENOBY=WARN NODATE NONUMBER LS=95 PS=54;
756
757 TITLE1 'Chapter 1 Example: A Simple SAS Job';
758
759 LIBNAME classlib 'P:\Teaching\BIOS-511\course-data';
NOTE: Libref CLASSLIB was successfully assigned as follows:
      Engine:          V9
      Physical Name: P:\Teaching\BIOS-511\course-data
760
761 DATA work.class;
762   SET classlib.class;
763
764   htmnet = ht*2.54;
765   wtmnet = wt/2.2;
766
767   LABEL htmnet = 'Height in Centimeters'
768         wtmnet = 'Weight in Kilograms';
769 RUN;

NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 765:13

NOTE: There were 6 observations read from the data set CLASSLIB.CLASS.
NOTE: The data set WORK.CLASS has 6 observations and 7 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

.
.
.

```

Figure 13: Simple SAS Program Partial Output

Chapter 1 Example: A Simple SAS Job
Listing of Data Portion of Data Set WORK.CLASS

Obs	NAME	SEX	AGE	HT	WT	htmet	wtmet
1	CHRISTIANSEN	M	37	71	195	180.34	88.6364
2	HOSKING J	M	31	70	160	177.80	72.7273
3	HELMS R	M	41	74	195	187.96	88.6364
4	PIGGY M	F	.	48	.	121.92	.
5	FROG K	M	3	12	1	30.48	0.4545
6	GONZO		14	25	45	63.50	20.4545

Chapter 1 Example: A Simple SAS Job
Listing of Descriptor Portion of Data Set WORK.CLASS

The CONTENTS Procedure

Data Set Name	WORK.CLASS	Observations	6
Member Type	DATA	Variables	7
Engine	V9	Indexes	0
Created	08/20/2014 10:01:44	Observation Length	56
Last Modified	08/20/2014 10:01:44	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_64		
Encoding	wlatin1 Western (Windows)		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Label
3	AGE	Num	8	
4	HT	Num	8	
1	NAME	Char	12	
2	SEX	Char	1	
5	WT	Num	8	
6	htmet	Num	8	Height in Centimeters
7	wtmet	Num	8	Weight in Kilograms

Syntax Errors

- SAS as you will use it in this class is very forgiving about syntax errors.
- The Windows SAS editor detects many errors as you type and highlights in red any syntax that it does not understand.
- If the code you submit does not work as you expected, it remains in the editor for you to make changes and resubmit until the code is working correctly.
- SAS error messages can sometimes be hard to interpret.
 - Always check first for common errors:
 1. Missing semi-colon (most common error)
 2. Misspelled variable or PROC name
 3. Unbalanced quotation marks
 4. Recreating a dataset that you have open for viewing
- When you determine the cause of a syntax error, you should commit the SAS log message to memory.
- No SAS log should ever be turned in with ERRORS or WARNINGS that are not fully understood.
- Having ERRORS in a homework/midterm/project/quiz SAS log will result in increasingly stronger penalties as the semester progresses.

Ways to run SAS

- You can start a SAS session by
 1. Entering the SAS command at a system prompt
 2. Clicking on a SAS shortcut icon
 3. Selecting SAS from the Start menu under Windows.
- Two modes of execution or environments are commonly used to run SAS programs:
 1. Interactive window environment
 - The SAS window environment is a collection of windows for editing and running programs, displaying the SAS log, displaying procedure output, and more.
 - The first lab of the course will demo the SAS Window Environment.
 - By default, the SAS log is not written to a permanent file.
 2. Non-interactive or batch mode
 - In batch mode, SAS programming statements are stored in a file, and the program runs when you issue a SAS command referencing the file.
 - In this mode, the SAS log and output are written to files and not displayed on the screen.
 - The SAS log from a SAS program run in batch mode is written to a file with the same filename as the source file and .log as the extension.
 - The output is written to a file with the same filename as the source file and either .html, .lst, .pdf, or .doc as the extension.
 - By default, the .log and output files are written to the current directory

- If an error occurs when running a SAS program in batch mode, the .log file will be created but possibly not the output file.
- On a PC where SAS is installed, you can right-click on a *.SAS file and choose “Batch Submit with SAS” from the action menu.
- Alternatively, batch SAS can be invoked with a SAS command.

Figure 14: Batch SAS command from windows command prompt

A screenshot of a Windows command prompt window. The command entered is: C:\Users\psioda> "C:\Program Files\SASHome\SASFoundation\9.4\SAS.EXE" exampleBinDesign-BatchSAS.sas _

```
C:\Users\psioda> "C:\Program Files\SASHome\SASFoundation\9.4\SAS.EXE" exampleBinDesign-BatchSAS.sas _
```

If interested, see the following (advanced) paper:

<http://analytics.ncsu.edu/sesug/2012/PO-08.pdf>

* Unfortunately the right-click action “Batch Submit with SAS” is not available on lab PCs because SAS is actually running on some other machine. It is available if you install SAS on your own PC or if you run SAS using the VCL (<https://vcl.unc.edu>).

- Most programmers who use SAS “for a living” develop the SAS programs in an interactive windowing environment and then use batch SAS to execute them once completed.