Introduction to SAS

All programs and data sets used for these notes are available from my course website. These files are gpa.sas, gpa.txt, gpa.csv, gpa_names.csv, cereal.sas, and cereal.csv.

Background

SAS is a very widely used statistical software package. The software was originally developed by students and professors at North Carolina State University in the 1960s and 1970s. These individuals eventually left the university and formed the SAS Institute based in the Research Triangle area of North Carolina. The SAS Institute is now the largest privately owned software company in the world. Originally, SAS stood for "Statistical Analysis System" but it is now just simply "SAS".

When I was going to graduate school, there were no real competitors. SAS by far was the most powerful statistical software package with respect to what it could do. All statistics graduate students learned how to use SAS. Software like SPSS was not used by statisticians but by some individuals in other fields, like the social sciences, that did not necessarily need the best software. Part of SAS's appeal was that SAS was cheap for those in academia. Once statistics graduate students received their degree, these students would ask their companies if they could use SAS. The SAS Institute would then charge thousands of dollars in yearly subscription fees per user license to these companies!

The statistical software environment began to change on February 29, 2000 when R version 1.0.0 was released. Unlike SAS, R is completely free. The growth of R has been huge to a point now that R is the predominant statistical software in Departments of Statistics. Still, SAS is used a lot in academia and in industry. Currently, I think students need to have a background in both

software packages. It will be interesting to see if this is still true in the future :).

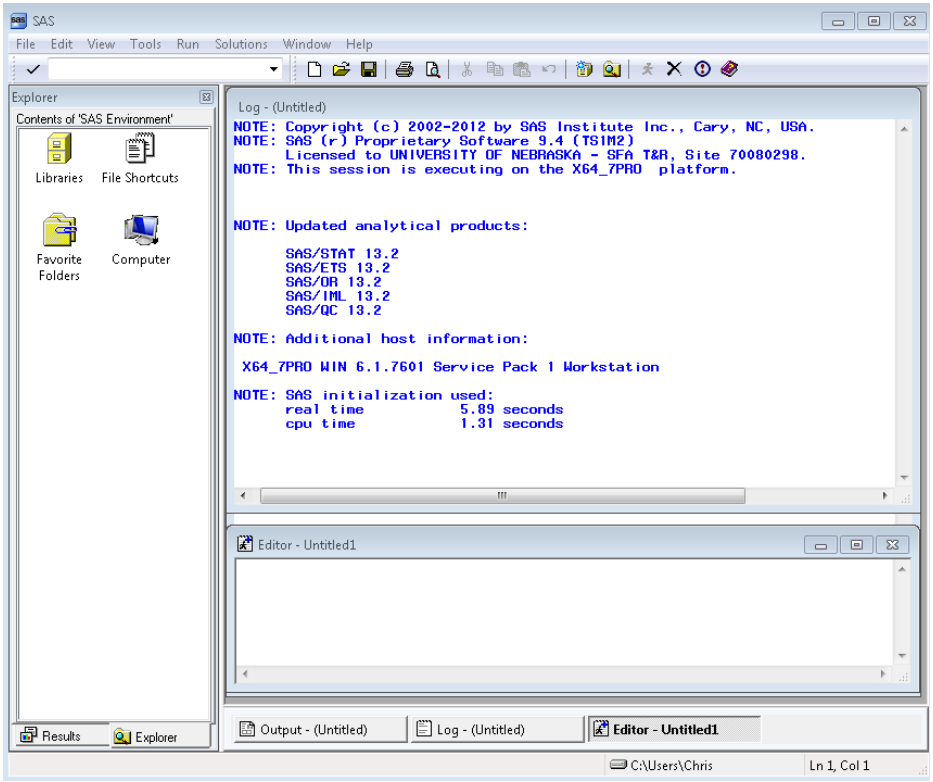
Installation

It is not necessarily easy for an individual to purchase SAS outside of academia! For example, go to the SAS website (`http://www.sas.com`) and you will notice that there is no "purchase here" link.

For all of UNL, the Department of Statistics is the distributor of SAS. Inidviduals can obtain SAS for a small fee from the department. The SAS installation on a computer is very large ($>$ 10GB), and there can be some issues that occur during installation. Note that SAS is also available on the computers in the two small computer rooms within our department as well as a few other computer rooms on campus.

Basics

Below is what SAS looks like when you open it.



There are a number of windows within SAS:

- Enhanced editor: Statisticians primarily write computer programs to perform statistical analyses. Programs for SAS are written here. This editor has syntax highlighting so that different colors are used for different types of code. Also, the editor enables code folding so that multiple lines of code can be "folded" into one line.
- Log: During a running of a program, information regarding the execution of code (e.g., syntax errors) will be printed here.
- Results Explorer: The output resulting from running a program (e.g., information about the fit of a regression model) will

appear here. By default, all output created is HTML-based and opened within this window.

- Output: This is where the output from procedures used to go by default (prior to SAS version 9.3). It still can be useful to use this window at times.
- Explorer/Results: Data sets are listed in the Libraries location of the Explorer tab. Links to specific output are in the Results tab.

These windows are all re-sizable and can be moved around. Important buttons on the main toolbar include:

- Submit : Select this button to run the whole or a highlighted portion of a program.
- Break : Select to stop a program while it is running.
- Save : Select to save the program.

All SAS programs have a .sas file extension. These files are simply plain text so they can be opened in programs such as Notepad. SAS is not case sensitive with its programs.

Simple program

My program gpa.sas is a very basic SAS program. It reads in a data set located in the files GPA.txt (plain text file using space delimiters, NOT tab delimiters) and gpa.csv (plain text file using comma delimiters). Both of these files contain the same 20 observations involving high school and college GPA of students. Our eventual purpose will be to use high school GPA to estimate college GPA with a regression model.

To open the SAS program, simply drag and drop it into the Enhanced Editor window or select FILE > OPEN. Below is a screen capture of the program in the Enhanced Editor.

gpa.sas

```
dm "log; clear; odsresults; clear;";
options linesize=64 nonumber nodate;

*********************************************************************;
* AUTHOR:  Chris Bilder                                             *;
* DATE:  4-22-16                                                    *;
* PURPOSE: Data analysis example using the gpa data set             *;
*********************************************************************;

*Will appear as the first line of every page of output;
title1 "Chris Bilder, STAT 850";


*Read in the data set from a space delimitted text file;
data set1;
  infile "C:\data\gpa.txt" firstobs=2;
  input HS College;
run;


*Read in the data set from a comma delimitted text file;
data set1;
  infile "C:\data\gpa.csv" firstobs=2 delimiter=",";
  input HS College ;
run;


title2 "The HS and College GPA data set";
proc print data=set1;
run;
```

Important aspects in the beginning of the program:

- Line 1: This clears any information that may be in the log window. If you also want to clear information in the Results Viewer (I usually do), include `odsresults; clear;` within the quotes. The `dm` code means "display manager".
- Line 2: Various options can be set here. I specify these options to help with producing my lecture notes. For example, the `linesize = 64` limits the number of characters on each line of output in the Results Viewer to be no larger than 64.

- Comments begin with a asterisk * and are color coded in green. Note that only one asterisk is actually needed at the beginning of the line. I use more than one at times because this is a standard way in SAS programs to make code more readable.
- The other code is colorized according to its purpose as well. This can be controlled by selecting TOOLS > OPTIONS > ENHANCED EDITOR > APPEARANCE.
- The `title1` statement gives information that will be printed on line 1 of every page in the Results Viewer.
- Semicolons end every complete statement in SAS!

Next in the program is a *datastep*. These are used to change information in a data set. In this case, it is used to read in the data from a file. Below is an explanation of the code:

- Line 1: All datasteps begin with `data`. The word after it gives the name of a data set to be created. This name needs to start with a letter and can include numbers and underscores.
- Line 2: The `infile` statement indicates where the data file is located. The `firstobs` option indicates to SAS to start reading information from the file beginning at line 2. This is used here here because the first line in the file contains variable names. If the file was comma delimited, the option `delimiter=","` could be added to this line of code.
- Line 3: The `input` statement declares the variable names. Again, these names need to start with a letter and can include numbers and underscores.
- Line 4: The `run` commands tells SAS to execute the code.
- Code folding is available by selecting the minus symbol ▫ next to the `data` line. This will put a plus symbol it is place. Selecting the plus will unfold the code.

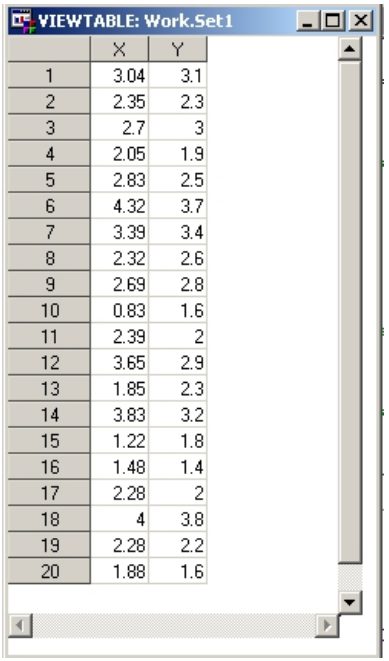
- Indenting of the code between `data` and `run` is standard practice to make code easier to read.

To run the code so far, I highlight from the beginning of the program to the end of the datastep and then select submit. This code produces no output itself. Below is what the Log window looks like after running this segment of code.

```
1 dm "log;clear;odsresults;clear;";
2 options ps=50 ls=75 pageno=1;
3
4 ************************************************;
5 * NAME:  Chris Bilder                           *;
6 * DATE:  4-22-16                                *;
7 * PURPOSE: Example with gpa data set            *;
8 ************************************************;
9
10   *Will appear as the first line of every page of output;
11   title1 'Chris Bilder, STAT 850';
12
13
14 *Read in the data set from a space delimitted text file;
15   data set1;
16     infile 'C:\data\gpa.txt' firstobs=2;
17     input HS College;
18   run;
NOTE: The infile 'C:\data\gpa.txt' is:
      Filename=C:\data\gpa.txt,
      RECFM=V,LRECL=32767,File Size (bytes)=213,
      Last Modified=22Apr2016:22:38:43,
      Create Time=22Feb2012:10:27:54
NOTE: 20 records were read from the infile 'C:\data\gpa.txt'.
      The minimum record length was 5.
      The maximum record length was 9.
NOTE: The data set WORK.SET1 has 20 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

The notes provided with the datastep indicate the code ran properly. In the Explorer window, the data set is shown in the

Work library (select LIBRARIES > WORK). This is the default location for SAS to put data sets. Selecting `set1` in the library opens the data set into a window.



	X	Y
1	3.04	3.1
2	2.35	2.3
3	2.7	3
4	2.05	1.9
5	2.83	2.5
6	4.32	3.7
7	3.39	3.4
8	2.32	2.6
9	2.69	2.8
10	0.83	1.6
11	2.39	2
12	3.65	2.9
13	1.85	2.3
14	3.83	3.2
15	1.22	1.8
16	1.48	1.4
17	2.28	2
18	4	3.8
19	2.28	2.2
20	1.88	1.6

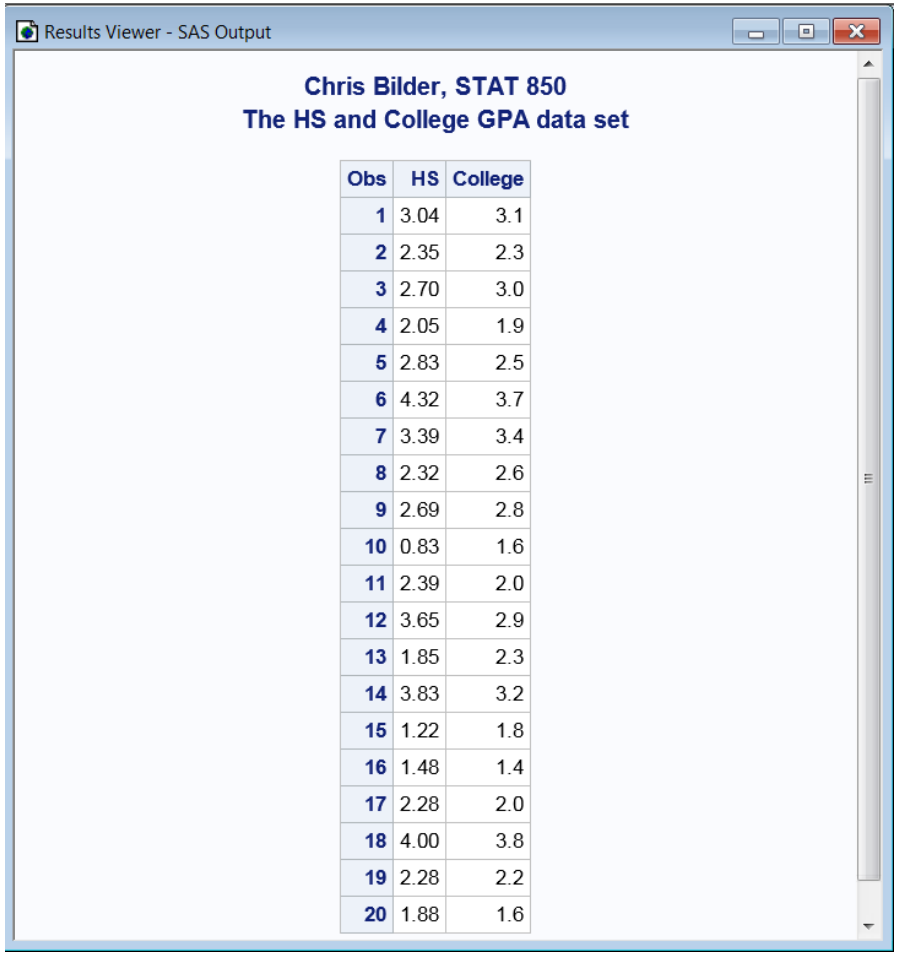
Comments:

- Data sets in the work library are actual files located on your computer, and they are stored in temporary folders that are deleted when SAS is closed (C:\Users\Chris\AppData\Local\Temp\SAS Temporary Files on my computer). We will see later how to create a permanent SAS data set in a user created library later.
- The data set needs to be closed before it can be modified again via code.

As mentioned earlier, SAS output is displayed in the Results Viewer. A simple way to put information into this window is

by printing the data set with `proc print`:

```
title2 'The HS and College GPA data set';
proc print data=set1;
run;
```



Obs	HS	College
1	3.04	3.1
2	2.35	2.3
3	2.70	3.0
4	2.05	1.9
5	2.83	2.5
6	4.32	3.7
7	3.39	3.4
8	2.32	2.6
9	2.69	2.8
10	0.83	1.6
11	2.39	2.0
12	3.65	2.9
13	1.85	2.3
14	3.83	3.2
15	1.22	1.8
16	1.48	1.4
17	2.28	2.0
18	4.00	3.8
19	2.28	2.2
20	1.88	1.6

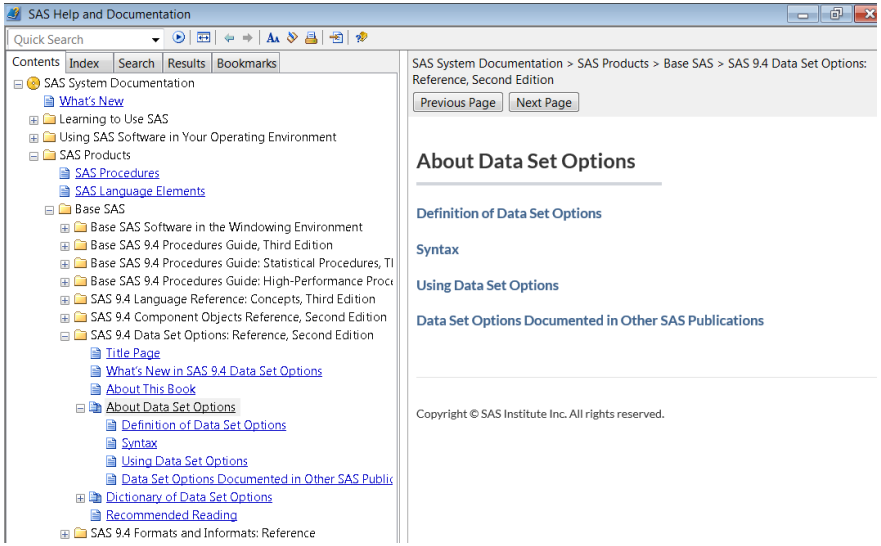
Below is an explanation of the code:

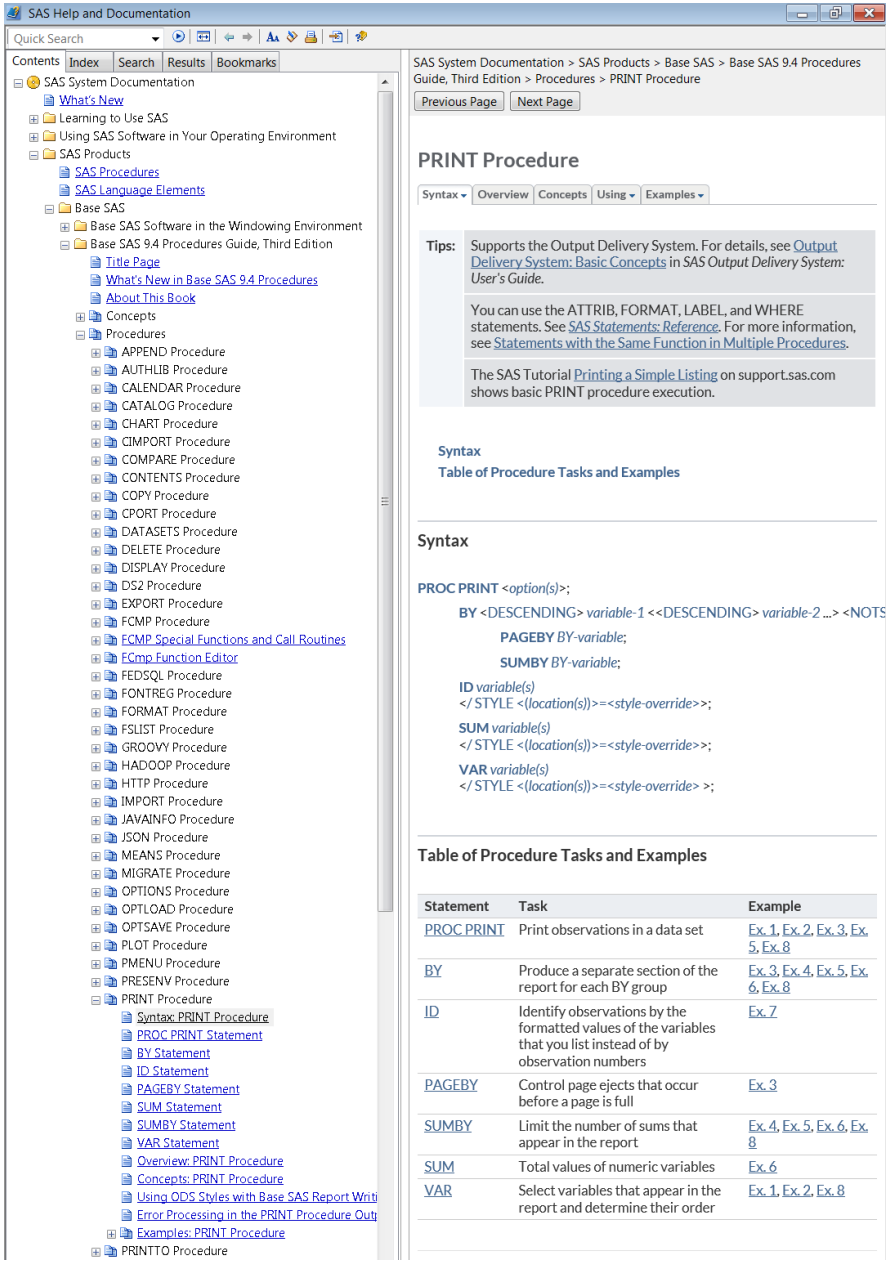
- A standard way to organize code and output is to use the second line of every page in the Results Viewer to indicate what is being displayed. I did this through a `title2` statement. Note that this statement can be located on other lines in the Enhanced Editor, but needs to be before the `run` statement.
- The `proc` line defines what procedure to run. For most procedures, there is a `data` option which allows one to specify the data set to be used. When a procedure needs a data set, one can get away with not specifying any with `data` because SAS assumes the last active data set is being used. I recommend against doing this to prevent programming errors!
- The `run` line simply runs the procedure.

Help

SAS consists of many different *products* which can be purchased separately. These products include: Base, STAT, GRAPH, ETS, IML, OR, QC, ... The datastep and `proc print` are part of Base. This product organization is helpful to know when finding "help" because SAS organizes its help system in this format.

One way to find help is to select the help icon on the toolbar. After selecting this icon, I found the help for a data step and `proc print` in SAS:


SAS Help and Documentation
Quick Search
Contents
Index
Search
Results
Bookmarks
SAS System Documentation
What's New
Learning to Use SAS
Using SAS Software in Your Operating Environment
SAS Products
SAS Procedures
SAS Language Elements
Base SAS
Base SAS Software in the Windowing Environment
Base SAS 9.4 Procedures Guide, Third Edition
Base SAS 9.4 Procedures Guide: Statistical Procedures, TI
Base SAS 9.4 Procedures Guide: High-Performance Proc
SAS 9.4 Language Reference: Concepts, Third Edition
SAS 9.4 Component Objects Reference, Second Edition
SAS 9.4 Data Set Options: Reference, Second Edition
Title Page
What's New in SAS 9.4 Data Set Options
About This Book
About Data Set Options
Definition of Data Set Options
Syntax
Using Data Set Options
Data Set Options Documented in Other SAS Public
Dictionary of Data Set Options
Recommended Reading
SAS 9.4 Formats and Informats: Reference
SAS System Documentation > SAS Products > Base SAS > SAS 9.4 Data Set Options: Reference, Second Edition
Previous Page
Next Page
About Data Set Options
Definition of Data Set Options
Syntax
Using Data Set Options
Data Set Options Documented in Other SAS Publications
Copyright © SAS Institute Inc. All rights reserved.


SAS Help and Documentation
Quick Search
Contents
Index
Search
Results
Bookmarks
SAS System Documentation
What's New
Learning to Use SAS
Using SAS Software in Your Operating Environment
SAS Products
SAS Procedures
SAS Language Elements
Base SAS
Base SAS Software in the Windowing Environment
Base SAS 9.4 Procedures Guide, Third Edition
Title Page
What's New in Base SAS 9.4 Procedures
About This Book
Concepts
Procedures
APPEND Procedure
AUTHLIB Procedure
CALENDAR Procedure
CATALOG Procedure
CHART Procedure
CIMPORT Procedure
COMPARE Procedure
CONTENTS Procedure
COPY Procedure
CPORT Procedure
DATASETS Procedure
DELETE Procedure
DISPLAY Procedure
DS2 Procedure
EXPORT Procedure
FCMP Procedure
FCMP Special Functions and Call Routines
FCmp Function Editor
FEDSQL Procedure
FONTREG Procedure
FORMAT Procedure
FSLIST Procedure
GROOVY Procedure
HADOOP Procedure
HTTP Procedure
IMPORT Procedure
JAVAINFO Procedure
JSON Procedure
MEANS Procedure
MIGRATE Procedure
OPTIONS Procedure
OPTLOAD Procedure
OPTSAVE Procedure
PLOT Procedure
PMENU Procedure
PRESENV Procedure
PRINT Procedure
Syntax: PRINT Procedure
PROC PRINT Statement
BY Statement
ID Statement
PAGEBY Statement
SUM Statement
SUMBY Statement
VAR Statement
Overview: PRINT Procedure
Concepts: PRINT Procedure
Using ODS Styles with Base SAS Report Writi
Error Processing in the PRINT Procedure Outp
Examples: PRINT Procedure
PRINTTO Procedure
SAS System Documentation > SAS Products > Base SAS > Base SAS 9.4 Procedures Guide, Third Edition > Procedures > PRINT Procedure
Previous Page
Next Page
PRINT Procedure
Syntax
Overview
Concepts
Using
Examples
Tips:
Supports the Output Delivery System. For details, see Output Delivery System: Basic Concepts in SAS Output Delivery System: User's Guide.
You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See SAS Statements: Reference. For more information, see Statements with the Same Function in Multiple Procedures.
The SAS Tutorial Printing a Simple Listing on support.sas.com shows basic PRINT procedure execution.
Syntax
Table of Procedure Tasks and Examples
Syntax
PROC PRINT <option(s)>;
BY <DESCENDING> variable-1 <<DESCENDING> variable-2 ...> <NOTS
PAGEBY BY-variable;
SUMBY BY-variable;
ID variable(s)
</ STYLE <(location(s))>=<style-override>>;
SUM variable(s)
</ STYLE <(location(s))>=<style-override>>;
VAR variable(s)
</ STYLE <(location(s))>=<style-override> >;
Table of Procedure Tasks and Examples
Statement
Task
Example
PROC PRINT
Print observations in a data set
Ex. 1, Ex. 2, Ex. 3, Ex. 5, Ex. 8
BY
Produce a separate section of the report for each BY group
Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 8
ID
Identify observations by the formatted values of the variables that you list instead of by observation numbers
Ex. 7
PAGEBY
Control page ejects that occur before a page is full
Ex. 3
SUMBY
Limit the number of sums that appear in the report
Ex. 4, Ex. 5, Ex. 6, Ex. 8
SUM
Total values of numeric variables
Ex. 6
VAR
Select variables that appear in the report and determine their order
Ex. 1, Ex. 2, Ex. 8

Below is a zoomed in version of part of the `proc print` help:

Syntax

PROC PRINT <*option(s)*>;

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* ...> <NOTSORTED>;

PAGEBY *BY-variable*;

SUMBY *BY-variable*;

ID *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override*>>;

SUM *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override*>>;

VAR *variable(s)*
</ STYLE <(*location(s)*)>=<*style-override*> >;

Table of Procedure Tasks and Examples

Statement	Task	Example
PROC PRINT	Print observations in a data set	Ex. 1, Ex. 2, Ex. 3, Ex. 5, Ex. 8
BY	Produce a separate section of the report for each BY group	Ex. 3, Ex. 4, Ex. 5, Ex. 6, Ex. 8
ID	Identify observations by the formatted values of the variables that you list instead of by observation numbers	Ex. 7
PAGEBY	Control page ejects that occur before a page is full	Ex. 3
SUMBY	Limit the number of sums that appear in the report	Ex. 4, Ex. 5, Ex. 6, Ex. 8
SUM	Total values of numeric variables	Ex. 6
VAR	Select variables that appear in the report and determine their order	Ex. 1, Ex. 2, Ex. 8

The syntax provides a way to understand what each portion of the code does along with examples. For example, the `var` statement allows one to select particular variables to print. Thus,

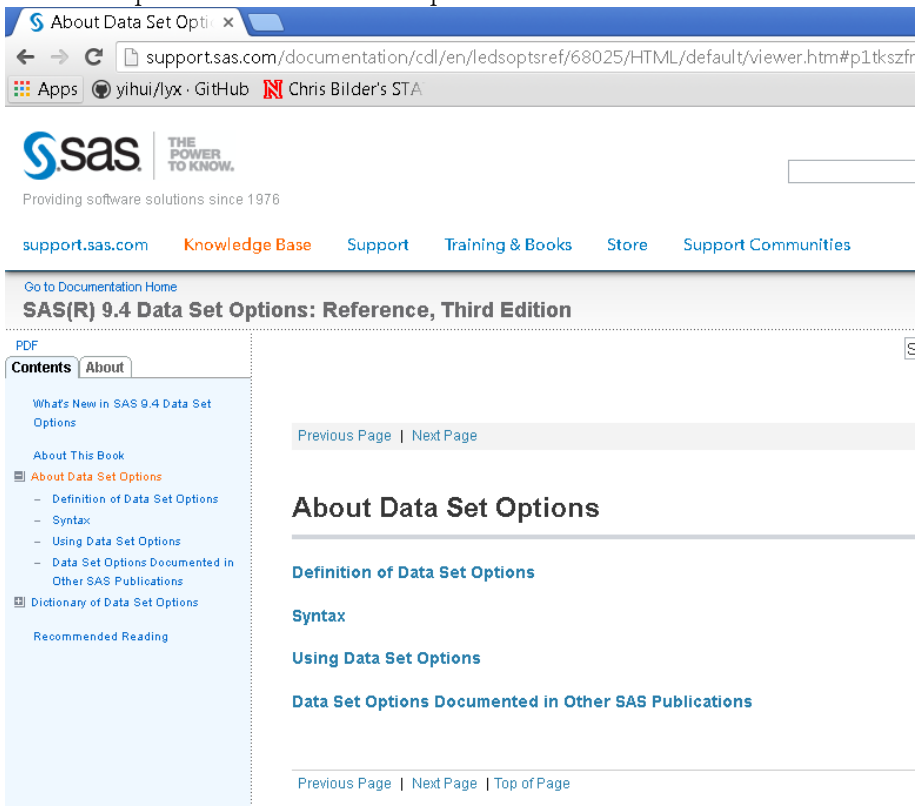
```
proc print data=set1;
```



```
  var HS;
run;
```

prints only the high school GPAs.

Other ways to find help are through web searches. Many web searches direct you to SAS's online documentation. Below is a screen capture for the datastep:



Import data into SAS

There are a number of file formats for data outside of SAS so there are a number of ways to get data into a SAS data set. Below are

some examples.

- Include data in the program: Rather than reading a small data set from an external file, it can be more simple to just include the data as part of the code.

```
data set1;
  input HS College;
  datalines;
  3.04 3.1
  2.35 2.3
  2.7 3.0
  2.55 2.45
  2.83 2.5
  4.32 3.7
  3.39 3.4
  2.32 2.6
  2.69 2.8
  2.83 3.6
  2.39 2.0
  3.65 2.9
  2.85 3.3
  3.83 3.2
  2.22 2.8
  1.98 2.4
  2.88 2.6
  4.0 3.8
  2.28 2.2
  2.88 2.6
;
run;
```

  Notes:

  – There are no semicolons after each line of data.
  – A semicolon cannot be put at the end of the last line of data! Instead, it needs to be on a new line after the data.
  – Some older SAS users may still use a `cards` statement rather than a `datalines` statement before entering the data.

- Mutiple observations can be read from the same line by using the double ampersand symbol @@

```
data set2;
  input HS College @@;
  datalines;
  3.04 3.1   2.35 2.3   2.7 3      2.55 2.45
  2.83 2.5   4.32 3.7   3.39 3.4   2.32 2.6
  2.69 2.8   2.83 3.6   2.39 2     3.65 2.9
  2.85 3.3   3.83 3.2   2.22 2.8   1.98 2.4
  2.88 2.6   4 3.8      2.28 2.2   2.88 2.6
;
run;
```

- Non-numeric (character) variables: When there are variables that include non-numerical values, one needs to include a dollar sign $ after the variable name.

```
data set3;
  infile "C:\data\gpa_names.csv" firstobs=2 delimiter=",";
  input HS College first_name $ last_name$;
run;
```

  There does not need to be a space between the variable name and $.

- Specific placement of data: The file format may dictate a very fine specification of which character positions contain values for particular variables. These positions can be specified with a single ampersand symbol @.

```
data set4;
  infile "C:\data\gpa_names.csv" firstobs=2 delimiter=",";
  input @1 HS @6 College;
run;
```

  One can also specify a range such as `HS 1-4` for the high school variable. This range format can be helpful when there are embedded blank spaces within a variable value.

- `proc import`: There are procedures available to import data

as well. Below is how I imported the comma delimited data set.

```
proc import out=set6 datafile="C:\data\gpa.csv" DBMS=CSV
   replace;
   getnames=yes;
   datarow=2;
run;
```

Note that the data set cannot be open in Excel when trying to import it.

The data set can be imported with `proc import` via point-and-click methods too by selecting FILE > IMPORT DATA to bring up the Import Wizard. In the first step, select the comma separated values format and then click on NEXT >. The second step allows you to browse to the file. The third step allows you to specify the library to put the data set (work library is o.k. for now). The last step allows you to specify a new SAS program to put the `proc import` code into in case you want to perform the importation with code in the future.

SAS is not perfect when importing data! Always remember to examine the data AFTER you read it in to make sure it is correct. Also, examine the log window for important statements about the results of the datastep. I provide two examples in my program that show when SAS does not import data correctly. Overall, I have found using a comma delimited format, rather than a space delimited format, helpful to prevent errors.

The data set can be made more descriptive through the use of labels. Below is a simple example.

```
data set3label;
  set set3;
  label HS = "High School GPA"
        College = "College GPA"
        first_Name = "First Name"
        last_Name = "Last Name";
run;
```


Notice how I used a previously created data set in the datastep with a `set` statement. This instructs SAS to create a new data set based on this previous one. We will use this type of syntax a lot!!! To see the inclusion of labels, open the data set in the work library.

Can Excel files be imported? Yes, but I recommend using a more simple comma delimitted format most of the time. With 32-bit and 64-bit computers/software, working directly with Excel files has become more difficult than it should be! If you still want to work directly with Excel files, I recommend using `proc import` to import the data into SAS.

Export data out of SAS

There are a number of ways to get data from a SAS data set into an external data file. Below are some examples.

- Datastep: The `outfile` and `put` statements are used with it.

```
data _null_;
  set set1;
  file 'C:\data\export1.csv' delimiter=",";
  put HS College;
run;
```

The `_null_` name for the data set is a way to not actually create a new data set for the operation. One could actually use something like `data set_new`, but this is not necessary.

- `proc export`: This procedure works similar to `proc import`.

```
proc export data=set1 outfile="C:\data\export2.csv" DBMS=CSV
   replace;
   putnames=yes;
run;
```

The data can also be exported via FILE > EXPORT DATA and following similar methods as with importing the data.

Figure 1: Cereal aisle at HyVee.

Again, one should check the log window and view the actual data file itself to make sure the data was correctly exported.

Cereal data

This examples reinforces some of the concepts learned earlier in this section and also shows new ways to use a datastep and a procedure.

A few years ago, I collected information on the nutritional content of dry cereals at a grocery store. This was done by first noting that one side of one aisle in many grocery stores usually contains all the cereals within a store. For example, Figure 1 shows what the cereal aisle used to look like in the HyVee at 5020 N 27th St. before its recent renovation. My research hypothesis was that there were different mean nutritional contents by shelf. For example, lower shelves may have more sugar content cereals than higher shelves.

The data used for this example was collected from a store a few years ago (not the HyVee in the picture). Note that there were only four shelves at this store and my sample size was 10 from

each shelf. Below is how I read the data into SAS and print the first five observations.

```
title1 "Chris Bilder, STAT 850";

proc import out=cereal datafile="C:\data\cereal.csv"
            DBMS=CSV replace;
  getnames=yes;
  datarow=2;
run;

title2 "Cereal data";
proc print data=cereal(obs=5);
run;
```

Chris Bilder, STAT 850
Cereal data

Obs	ID	Shelf	Cereal	size_g	sugar_g	fat_g	sodium_mg
1	1	1	Kellog's Razzle Dazzle Rice Crispies	28	10	0	170
2	2	1	Post Toasties Corn Flakes	28	2	0	270
3	3	1	Kellogg's Corn Flakes	28	2	0	300
4	4	1	Food Club Toasted Oats	32	2	2	280
5	5	1	Frosted Cheerios	30	13	1	210

There are many options that one can use with a data set by specifying them within parantheses right after the data set name. The previous code shows how only the first five observations of the cereal data set are used by `proc print`. Other options include `drop` and `keep` to specify which variables to use (see program for an example). While all of the data is not displayed here, the shelves are numbered from lowest (1) to highest (4).

We need to adjust the nutritional content variables (`sugar_g`, `fat_g`, and `sodium_g`) for the serving size because cereal boxes tend to have different serving sizes. Below is how I make the adjustment by a datastep.

```
data set1;
  set cereal;
  sugar = sugar_g/size_g;  *sugar content per cereal divided by
```

```
    serving size;
  fat = fat_g/size_g;
  sodium = sodium_mg/size_g;
  *remove the old variables below from the data set;
  keep ID Shelf Cereal sugar fat sodium;
run;

title2 "Cereal data adjusted for serving size";
proc print data=set1(obs = 5);
run;
```

Chris Bilder, STAT 850
Cereal data adjusted for serving size

Obs	ID	Shelf	Cereal	sugar	fat	sodium
1	1	1	Kellog's Razzle Dazzle Rice Crispies	0.35714	0.000000	6.0714
2	2	1	Post Toasties Corn Flakes	0.07143	0.000000	9.6429
3	3	1	Kellogg's Corn Flakes	0.07143	0.000000	10.7143
4	4	1	Food Club Toasted Oats	0.06250	0.062500	8.7500
5	5	1	Frosted Cheerios	0.43333	0.033333	7.0000

There are a number of statements in `proc print` which can make printing of output nicer. Below is how I use the `where` and `var` statements for illustrative purposes. I also remove the observations numbers with the `noobs` option.

```
proc print data=set1 noobs;
  where shelf=1;
  var shelf cereal sugar;
run;
```


Chris Bilder, STAT 850
Shelf #1 of the cereal data set

Shelf	Cereal	sugar
1	Kellog's Razzle Dazzle Rice Crispies	0.35714
1	Post Toasties Corn Flakes	0.07143
1	Kellogg's Corn Flakes	0.07143
1	Food Club Toasted Oats	0.06250
1	Frosted Cheerios	0.43333
1	Food Club Frosted Flakes	0.35484
1	Capn Crunch	0.44444
1	Capn Crunch's Peanut Butter Crunch	0.33333
1	Post Honeycomb	0.37931
1	Food Club Crispy Rice	0.06061

The `where` statement can be used with most procedures and has additional flexibility. For example, the use of `where shelf = 1 and sugar < 0.1` with `proc print` will print just low sugar cereals on the first shelf. Also, a similar use of this option can be done within the `proc print` line with `data=set1(where=(shelf = 1 and sugar < 0.1))`.

Sorting a data set can be useful to see aspects of the data which otherwise may be more difficult to detect without it. Also, some procedures may require that data be sorted prior to their use. Below is how `proc sort` can be used for this purpose.

```
proc sort data=set1;
  by shelf sugar;
run;
```

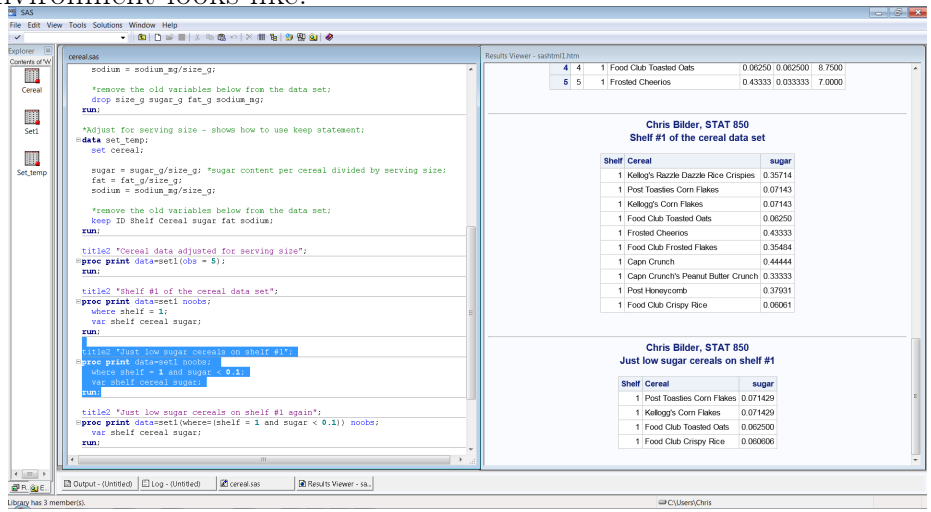
No output will be generated. Rather the data set will be rearrnaged in the work library.

VIEWTABLE: Work.Set1

	ID	Shelf	Cereal	sugar	fat	sodium
1	10	1	Food Club Crispy Rice	0.0606060606	0	10
2	4	1	Food Club Toasted Oats	0.0625	0.0625	8.75
3	2	1	Post Toasties Corn Flakes	0.0714285714	0	9.6428571429
4	3	1	Kellogg's Corn Flakes	0.0714285714	0	10.714285714
5	8	1	Capn Crunch's Peanut Butter Crunch	0.3333333333	0.0925925926	7.4074074074
6	6	1	Food Club Frosted Flakes	0.3548387097	0	5.8064516129
7	1	1	Kellog's Razzle Dazzle Rice Crispies	0.3571428571	0	6.0714285714
8	9	1	Post Honeycomb	0.3793103448	0.0172413793	7.5862068966
9	5	1	Frosted Cheerios	0.4333333333	0.0333333333	7
10	7	1	Capn Crunch	0.4444444444	0.0555555556	7.4074074074
11	11	2	Rice Crispies Treats	0.3	0.05	6.3333333333
12	18	2	Food Club Toasted Oats	0.303030303	0.0454545455	4.5454545455

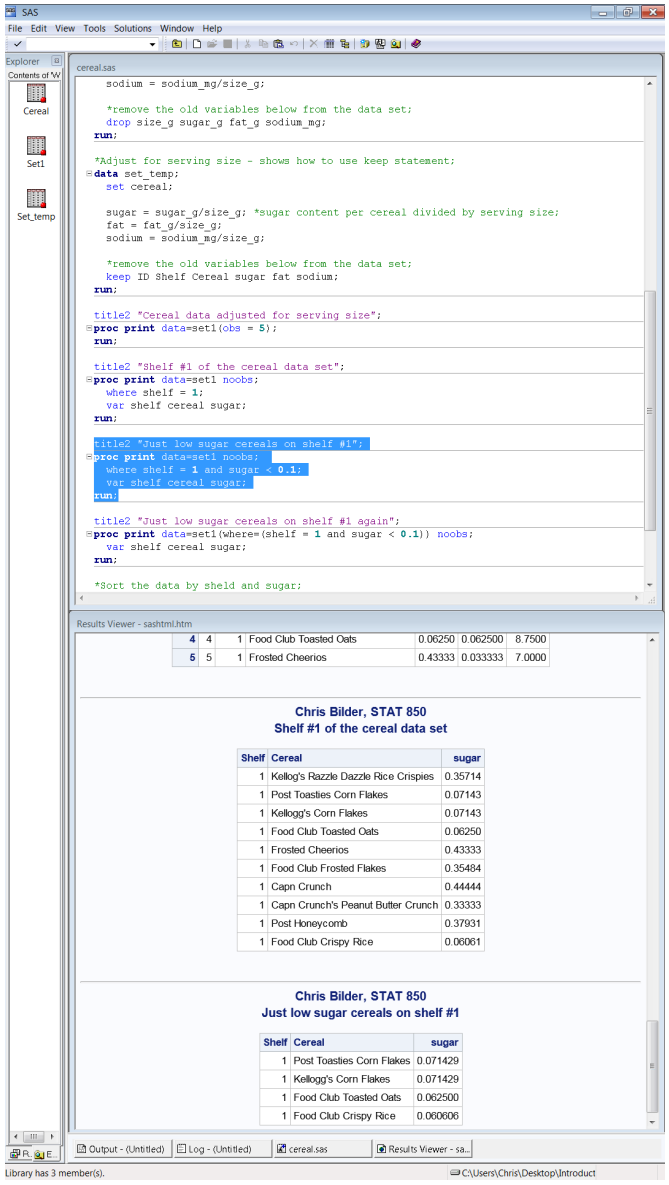
My SAS set-up

How do I work with SAS on my computer? If I am only using a single laptop monitor, this is what a screen capture of my working environment looks like:



I try to have two main windows of SAS available at all times. The log window is not shown because its is exactly the same size as the Results Viewer and behind it.

When I am in my office, I take advantage of my portrait oriented monitors and use the following work environment:


SAS
File Edit View Tools Solutions Window Help
Explorer
Contents of 'W
Cereal
Set1
Set_temp
cereal.sas
sodium = sodium_mg/size_g;
*remove the old variables below from the data set;
drop size_g sugar_g fat_g sodium_mg;
run;
*Adjust for serving size - shows how to use keep statement;
data set_temp;
set cereal;
sugar = sugar_g/size_g; *sugar content per cereal divided by serving size;
fat = fat_g/size_g;
sodium = sodium_mg/size_g;
*remove the old variables below from the data set;
keep ID Shelf Cereal sugar fat sodium;
run;
title2 "Cereal data adjusted for serving size";
proc print data=set1(obs = 5);
run;
title2 "Shelf #1 of the cereal data set";
proc print data=set1 noobs;
where shelf = 1;
var shelf cereal sugar;
run;
title2 "Just low sugar cereals on shelf #1";
proc print data=set1 noobs;
where shelf = 1 and sugar < 0.1;
var shelf cereal sugar;
run;
title2 "Just low sugar cereals on shelf #1 again";
proc print data=set1(where=(shelf = 1 and sugar < 0.1)) noobs;
var shelf cereal sugar;
run;
*Sort the data by sheld and sugar;
Results Viewer - sashtml.htm
4 4 1 Food Club Toasted Oats 0.06250 0.062500 8.7500
5 5 1 Frosted Cheerios 0.43333 0.033333 7.0000
Chris Bilder, STAT 850
Shelf #1 of the cereal data set
Shelf Cereal sugar
1 Kellog's Razzle Dazzle Rice Crispies 0.35714
1 Post Toasties Corn Flakes 0.07143
1 Kellogg's Corn Flakes 0.07143
1 Food Club Toasted Oats 0.06250
1 Frosted Cheerios 0.43333
1 Food Club Frosted Flakes 0.35484
1 Capn Crunch 0.44444
1 Capn Crunch's Peanut Butter Crunch 0.33333
1 Post Honeycomb 0.37931
1 Food Club Crispy Rice 0.06061
Chris Bilder, STAT 850
Just low sugar cereals on shelf #1
Shelf Cereal sugar
1 Post Toasties Corn Flakes 0.071429
1 Kellogg's Corn Flakes 0.071429
1 Food Club Toasted Oats 0.062500
1 Food Club Crispy Rice 0.060606
Output - (Untitled)
Log - (Untitled)
cereal.sas
Results Viewer - sa...
Library has 3 member(s).
C:\Users\Chris\Desktop\Introduct