

STAT604

Lesson SAS 17

Portions Copyright © 2009 SAS Institute Inc., Cary, NC, USA. All rights reserved. Reproduced with permission of SAS Institute Inc., Cary, NC, USA. SAS Institute Inc. makes no warranties with respect to these materials and disclaims all liability therefor.

Chapter 6: Debugging Techniques



6.1 Using the PUTLOG Statement

6.2 Using the DEBUG Option

Chapter 6: Debugging Techniques

6.1 Using the PUTLOG Statement

6.2 Using the DEBUG Option

Objectives

- Use the PUTLOG statement in the DATA step to help identify logic errors.

Business Scenario

A mailing list application at Orion Star is not working properly. Use debugging techniques to identify and correct the problem.



Syntax Errors versus Logic Errors

- A *syntax error* occurs when program statements do not conform to the rules of the SAS language.
 - An error message is written to the log.
- A *logic error* occurs when the program statements follow the rules but the results are not correct.
 - No notes are written to the log, so logic errors are often difficult to detect.

This section focuses on identifying **logic** errors.

Mailing List Application

Using the `orion.mailing_list` data set as input, create a new data set, `us_mailing`, that includes only those observations with a United States address.

The new data set should include only the person's name, street address, city, state, and ZIP code.

Browse the Input Data

Partial Listing of `orion.mailing_list`

Name	Address1	Address3
Abbott, Ray	2267 Edwards Mill Rd	Miami-Dade, FL, 33135, US
Aisbitt, Sandy	30 Bingera Street	Melbourne, 2001, AU
Akinfolarin, Tameaka	5 Donnybrook Rd	Philadelphia, PA, 19145, US
Amos, Salley	3524 Calico Ct	San Diego, CA, 92116, US
Anger, Rose	744 Chapwith Rd	Philadelphia, PA, 19142, US
Anstey, David	939 Hilltop Needmore Rd	Miami-Dade, FL, 33157, US
Antonini, Doris	681 Ferguson Rd	Miami-Dade, FL, 33141, US
Apr, Nishan	105 Brack Penny Rd	San Diego, CA, 92071, US
Ardskin, Elizabeth	701 Glenridge Dr	Miami-Dade, FL, 33177, US
Areu, Jeryl	265 Fyfe Ct	Miami-Dade, Fl, 33133, US
Arizmendi, Gilbert	379 Englehardt Dr	San Diego, CA, 91950, US
Armant, Debra	10398 Crown Forest Ct	San Diego, CA, 92025, US
Armogida, Bruce	1914 Lansing St	Philadelphia, PA, 19119, US
Arruza, Fauver	265 Fyfe Ct	Miami-Dade, FL, 33133, US
Asta, Wendy	3565 Lake Park Dr	Philadelphia, PA, 19145, US
Atkins, John	6137 Blue Water Ct	Miami-Dade, FL, 33161, US
Bahlman, Sharon	24 LaTrobe Street	Sydney, 2165, AU

Desired Results

Partial Listing of `us_mailing`

Name	Address1	City	State	Zip
Ray Abbott	2267 Edwards Mill Rd	Miami-Dade	FL	33135
Tameaka Akinfolarin	5 Donnybrook Rd	Philadelphia	PA	19145
Salley Amos	3524 Calico Ct	San Diego	CA	92116
Rose Anger	744 Chapwith Rd	Philadelphia	PA	19142
David Anstey	939 Hilltop Needmore Rd	Miami-Dade	FL	33157
Doris Antonini	681 Ferguson Rd	Miami-Dade	FL	33141
Nishan Apr	105 Brack Penny Rd	San Diego	CA	92071
Elizabeth Ardskin	701 Glenridge Dr	Miami-Dade	FL	33177
Jeryl Areu	265 Fyfe Ct	Miami-Dade	FL	33133
Gilbert Arizmendi	379 Englehardt Dr	San Diego	CA	91950
Debra Armant	10398 Crown Forest Ct	San Diego	CA	92025

The Current Program

```
data us_mailing;  
  set orion.mailing_list;  
  drop Address3;  
  length City $ 25 State $ 2 Zip $ 5;  
  if find(Address3,'US');  
  Name=catx(' ',  
            scan(Name,2,' ',''),  
            scan(Name,1,' ',''));  
  City=scan(Address3,1,' ','');  
  State=scan(address3,2,' ','');  
  Zip=scan(Address3,3,' ','');  
run;  
proc print data=us_mailing noobs;  
  title 'Current Output from Program';  
run;
```

Poll

Quiz



6.01 Quiz

Open and submit **p206a01**. What errors are in the output?

```
data us_mailing;
  set orion.mailing_list;
  drop Address3;
  length City $ 25 State $ 2 Zip $ 5;
  if find(Address3,'US');
  Name=catx(' ',
            scan(Name,2,' ',' '),
            scan(Name,1,' ',' '));
  City=scan(Address3,1,' ',' ');
  State=scan(address3,2,' ',' ');
  Zip=scan(Address3,3,' ',' ');
run;
proc print data=us_mailing noobs;
  title 'Current Output from Program';
run;
```

6.01 Quiz – Correct Answer

The values of **State** and **Zip** are incorrect.

Current Output from Program

Name	Address1	City	State	Zip
Ray Abbott	2267 Edwards Mill Rd	Miami-Dade	F	3313
Tameaka Akinfolarin	5 Donnybrook Rd	Philadelphia	P	1914
Salley Amos	3524 Calico Ct	San Diego	C	9211
Rose Anger	744 Chapwith Rd	Philadelphia	P	1914
David Anstey	939 Hilltop Needmore Rd	Miami-Dade	F	3315
Doris Antonini	681 Ferguson Rd	Miami-Dade	F	3314
Nishan Apr	105 Brack Penny Rd	San Diego	C	9207
Elizabeth Ardskin	701 Glenridge Dr	Miami-Dade	F	3317
Jeryl Areu	265 Fyfe Ct	Miami-Dade	F	3313
Gilbert Arizmendi	379 Englehardt Dr	San Diego	C	9195
Debra Armant	10398 Crown Forest Ct	San Diego	C	9202

The PUTLOG Statement

The PUTLOG statement can be used in the DATA step to

- display messages in the log
- display the value(s) of one or more variables.

General form of the PUTLOG statement:

```
PUTLOG <specifications>;
```

There are various ways to write the *specifications*.

Use PUTLOG to Write Text to the Log

To write text to the log, use this form of the PUTLOG statement:

```
PUTLOG 'text';
```

For example,

```
putlog 'Looking for country';
```

writes **Looking for country** to the log.

Use PUTLOG to Write the Value of a Variable

To write the name and value of a variable to the log, use this form of the PUTLOG statement:

```
PUTLOG variable-name=;
```

For example, if the value of the variable **City** is San Diego, the statement

```
putlog City=;
```

writes **City=San Diego** to the log.

Use PUTLOG to Write Formatted Values

To write the formatted value of a variable to the log, use this form of the PUTLOG statement:

```
PUTLOG variable-name format-namew.;
```

For example, if the value of the variable **City** is Philadelphia with a leading space, the statement

```
putlog City $quote22.;
```

writes “**Philadelphia**” to the log.



The value of *w* should be wide enough to display the value of the variable and the quotation marks.

Use PUTLOG to Write Values of All Variables

To write the current contents of the program data vector (PDV) to the log, use this form of the PUTLOG statement:

```
PUTLOG _ALL_;
```

Partial SAS Log

```
Name=Abbott, Ray Address1=2267 Edwards Mill Rd Address3=Miami-  
Dade, FL, 33135, US City= State= Zip= _ERROR_=0 _N_=1  
Name=Aisbitt, Sandy Address1=30 Bingera Street  
Address3=Melbourne, 2001, AU City= State= Zip= _ERROR_=0  
_N_=2  
Name=Akinfolarin, Tameaka Address1=5 Donnybrook Rd  
Address3=Philadelphia, PA, 19145, US City= State= Zip=  
_ERROR_=0 _N_=3
```

Special Variables

The temporary variables `_N_` and `_ERROR_` can be helpful when you debug a DATA step.

Variable	Description	Debugging Use
<code>_N_</code>	The number of times that the DATA step iterated	Display debugging messages for some number of iterations of the DATA step
<code>_ERROR_</code>	Initialized to 0, set to 1 when an error occurs	Display debugging messages when an error occurs.

Poll

Quiz



6.02 Quiz

Open the file **p206a02**. Insert statements to display the values of `_N_` and `_ERROR_` in the first three iterations of the DATA step. Submit and view the log.

```
data _null_;  
    set orion.donate;  
run;
```

6.02 Quiz – Correct Answer

Open the file **p206a02**. Insert statements to display the values of `_N_` and `_ERROR_` in the first three iterations of the DATA step.

Solution

```
data _null_;  
    set orion.donate;  
    if _n_ <= 3 then  
        putlog _n_ = _error_ =;  
run;
```

Alternate solution

```
data _null_;  
    set orion.donate (obs=3);  
    putlog _n_ = _error_ =;  
run;
```

Partial SAS Log

```
_N_=1  _ERROR_=0  
_N_=2  _ERROR_=0  
_N_=3  _ERROR_=0
```

Use `_N_` to execute PUTLOG on the first 3 iterations of the DATA step, or use `OBS=` to process only 3 observations.

The END= Option

The END= option creates a temporary variable that acts as an end-of-file indicator.

- The option can be used in SET and INFILE statements.
- The variable is initialized to 0 and is set to 1 when the last observation or record is read.

General form of the END= option:

```
SET SAS-data-set END=variable <options>;
```

```
INFILE 'raw-data-file' END=variable <options>;
```

Processing at the End of a DATA Step

Use conditional logic to check the value of the END= variable to determine if it is the last iteration of the DATA step.

```
data work.donate;  
    set orion.donate end=last;  
    <additional SAS statements>  
    if last=1 then do;  
        <additional SAS statements>  
    end;  
run;
```

The IF statement can also be written in this form:

```
if last then do;
```


Using PUTLOG with Conditional Logic

The program below displays a message in the log on the first iteration of the DATA step, and displays the contents of the PDV in the last iteration of the DATA step.

```
data _null_;  
    set orion.donate end=lastObs;  
    if _n_=1 then  
        putlog 'First iteration';  
    if lastObs then do;  
        putlog 'Final values of variables:';  
        putlog _all_;  
    end;  
run;
```

Partial SAS Log

```
First iteration  
Final values of variables:  
lastObs=1 Employee_ID=12447 Qtr=4 Amount=35 _ERROR_=0 _N_=16
```



Determining Logic Errors

p206d02

This demonstration illustrates using the PUTLOG statement to identify logic errors.

Chapter 6: Debugging Techniques



6.1 Using the PUTLOG Statement

6.2 Using the DEBUG Option

Objectives

- Use the DEBUG option in the DATA statement to identify logic errors.
- Set breakpoints in the debugger.

The DEBUG Option

The DEBUG option is an interactive interface to the DATA step. Commands are available to

- execute a DATA step one statement at a time
- examine the value of one or more variables
- watch one or more variables as they change in value.

General form of the DEBUG option:

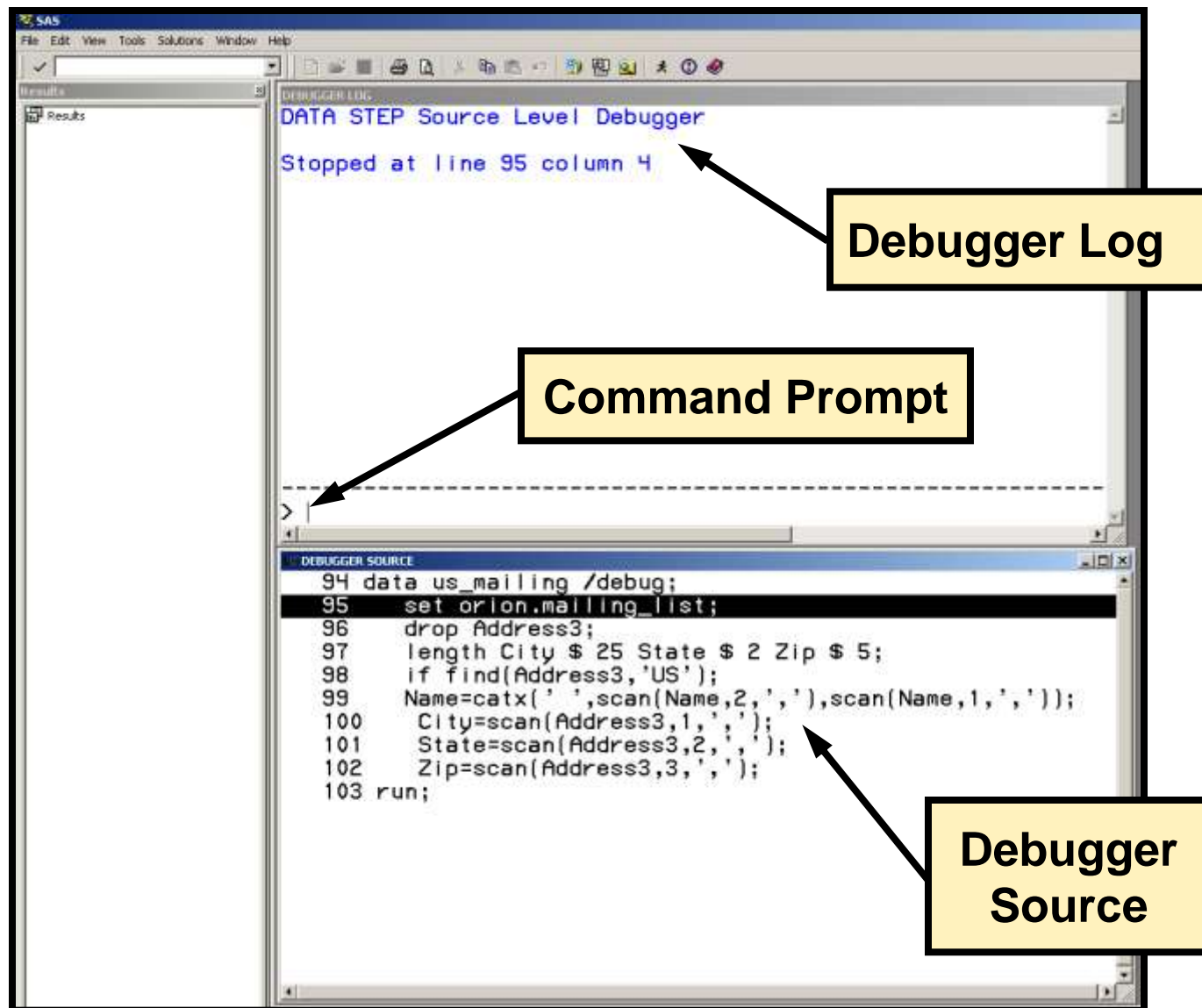
```
DATA data-set-name / DEBUG;
```

DATA Step Debugger

The debugger can only be used with a DATA step, not a PROC step.

- The DEBUG option is only available in interactive mode.
- A DATA step cannot be restarted after it ends; however, the final values of variables can be examined.

The DATA Step Debugger Window



DEBUG Commands

Common commands used with the DEBUG option:

Command	Alias	Action
STEP	ENTER key	executes statements one at a time
EXAMINE	E variable(s)	displays the value of one or more variables
WATCH	W variable(s)	suspends execution when the value of a watched variable changes
QUIT	Q	terminates a debugger session

DEBUG Commands

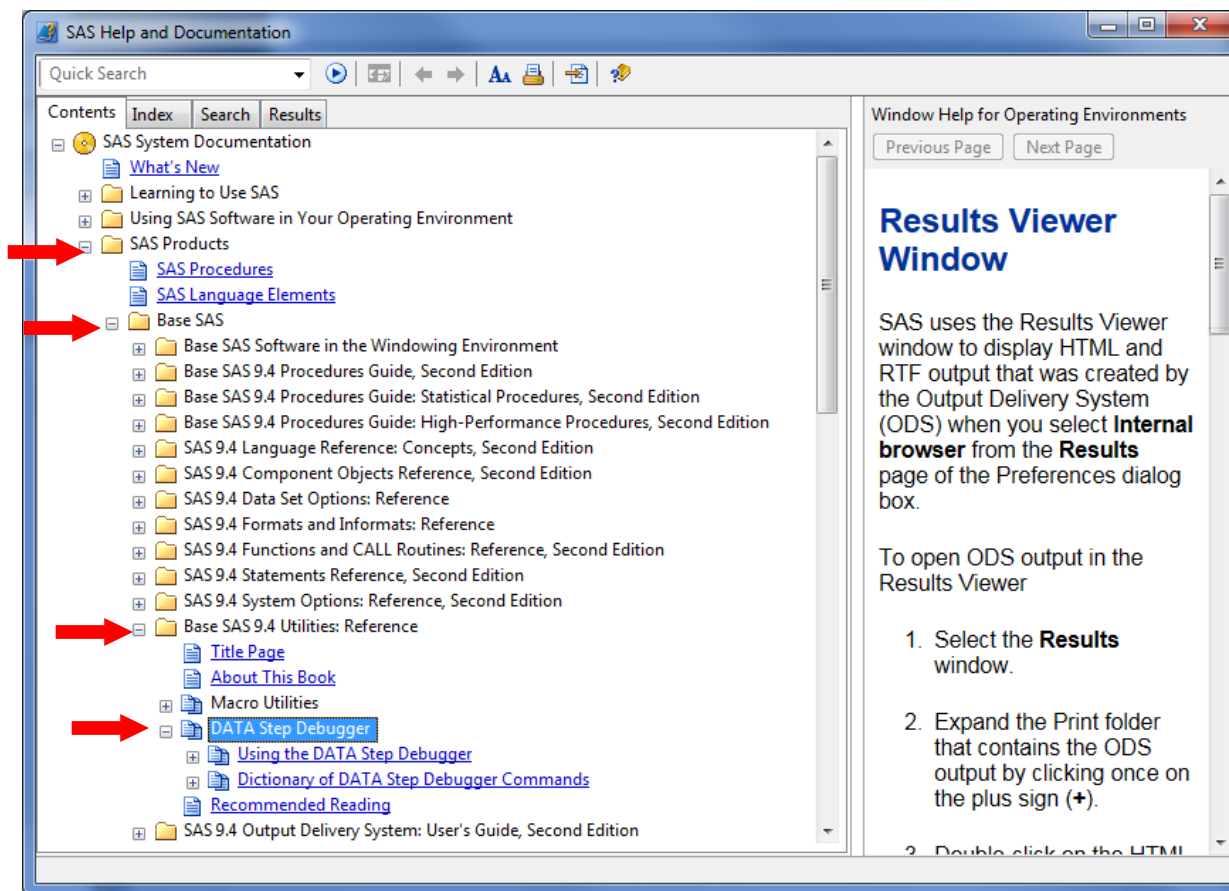
Other useful commands used with the DEBUG option:

Command	Alias	Action
LIST	L argument	displays all occurrences of the item listed in the argument L W – list watched variables
DELETE Watch	D W variable(s)	deletes the watch status of the listed variables
SET	SET	assigns a new value to the specified variable: SET variable=expression

DATA Step Debugger Documentation

For help on the DATA Step Debugger, select

SAS Products ⇒ **Base SAS** ⇒ **Base SAS 9.4 Utilities: Reference**
⇒ **DATA Step Debugger**.



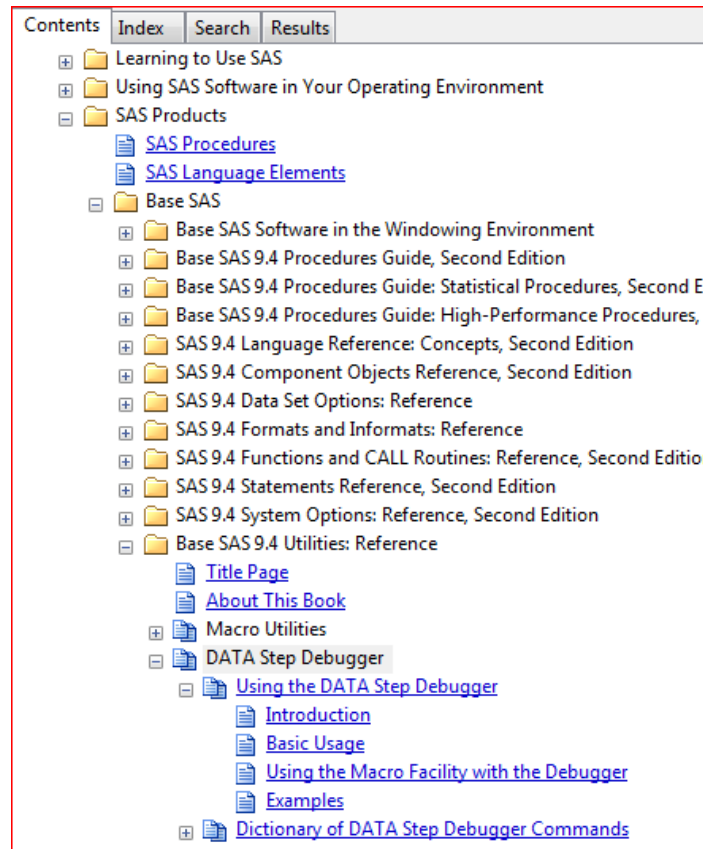
Poll

Quiz



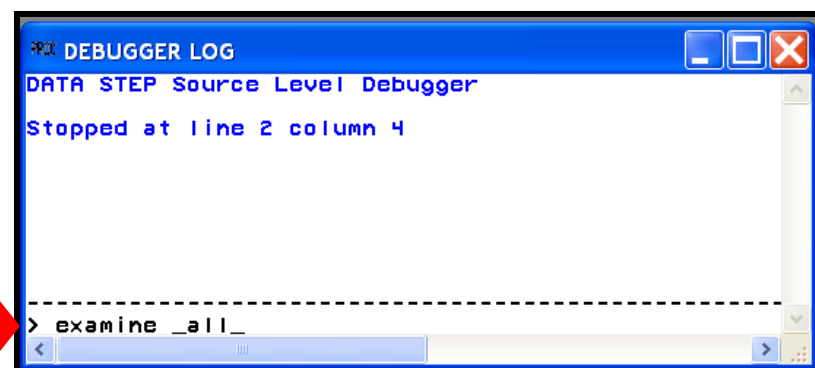
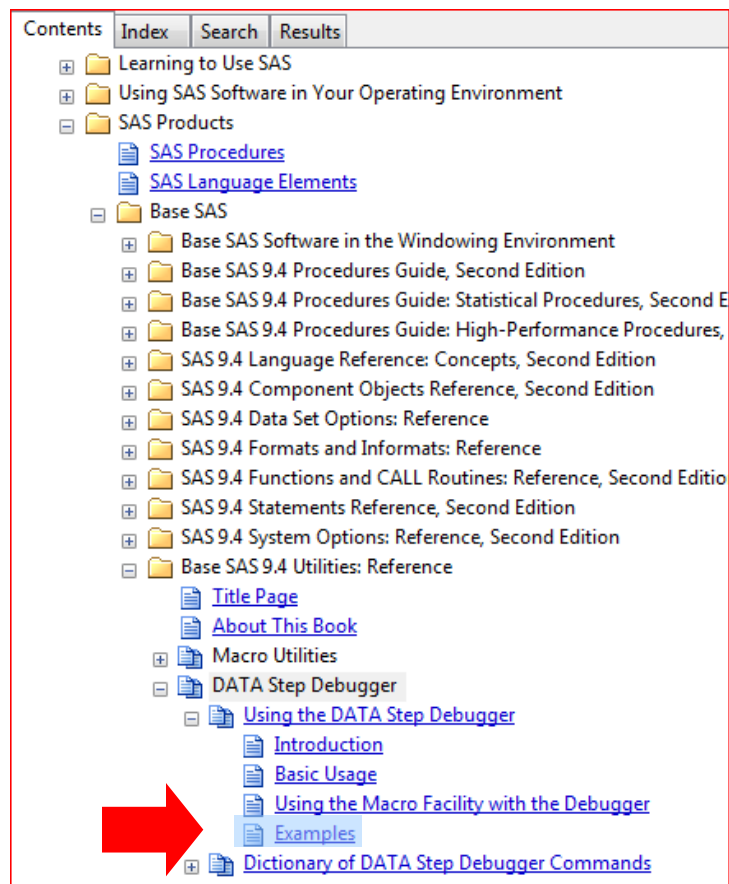
6.03 Quiz

Use the SAS Help facility to locate DATA Step Debugger examples. In the first example, what command is used to examine data values after the first iteration?



6.03 Quiz – Correct Answer

In the first example, what command is used to examine data values after the first iteration? **examine _all_**





Determining Logic Errors

p206d03

This demonstration uses the DATA step debugger to identify logic errors.

Setting Breakpoints

A *breakpoint* is a point at which program execution is suspended, allowing the user to submit debugger commands.

- Breakpoints can be conditional or unconditional.
- Breakpoints can be in effect on every iteration of the DATA step or after a set number of iterations.
- An exclamation mark appears to the left of the line in the Debugger Source window.

 Breakpoints are useful in DATA steps that iterate many times.

The BREAK Command

General form of the BREAK command:

BREAK *location* <AFTER *count*> <WHEN *expression*> <DO *group*>

location

The number of the line at which to suspend execution.

AFTER *count*

Break after the line has been executed *count* times.

WHEN *expression*

Break when *expression* evaluates to TRUE.

DO *group*

One or more debugger commands enclosed by a DO and an END statement.

Sample BREAK Commands

Command	Description
B 32	Set a breakpoint on line 32
B 25 after 4	Break on line 25 after it executed four times
B 7 when num=0	Break on line 7 when num=0
B 7 after 3 when num=0	Break on line 7 after it executed three times only if num=0
B 9 do; e city zip; end;	Break on line 9 and display the values of City and Zip automatically
D B 9	Delete the breakpoint on line 9

The GO Command

The GO command resumes program execution.

General form of the GO command:

GO <line-number>

line-number

The number of the program line at which execution is to be suspended

Sample GO Commands

Command	Description
G	Resume execution at the next executable line.
G 30	Resume execution and suspend execution at line 30.



Setting Breakpoints

This demonstration illustrates the use of breakpoints in the interactive debugger.

Chapter 10: Other SAS Languages



10.1 An Overview of Other Languages

10.2 Using the SQL Procedure

10.3 The SAS Macro Language

Chapter 10: Other SAS Languages

10.1 An Overview of Other Languages

10.2 Using the SQL Procedure

10.3 The SAS Macro Language

Objectives

- Describe other languages available in SAS.

Other SAS Languages

The languages that are available in SAS include the following:

- the SAS language
- SQL
- macro
- SCL
- SAS/C

Until now the focus was on the SAS language.

This chapter introduces SQL and the macro language.

Why Learn SQL?

SQL (Structured Query Language) is a standardized language used by many software products.

The SQL procedure in SAS

- can be used to retrieve, join, and update data in tables
- can perform more efficient join operations than DATA step merges in some cases
- can replace the need for multiple DATA and PROC steps with one query.

Why Learn Macro?

The SAS macro language permits code substitution as well as automatic code generation and execution.

Macro programming makes more efficient use of a SAS developer's time by

- simplifying program maintenance
- generating flexible, customizable code
- executing code iteratively.

Chapter 10: Other SAS Languages



10.1 An Overview of Other Languages

10.2 Using the SQL Procedure (Self-Study)

10.3 The SAS Macro Language

The SQL Procedure

The SQL procedure enables you to write ANSI standard SQL queries in a SAS program.

With PROC SQL you can access the following:

- SAS data sets
- other database data, with the appropriate SAS/ACCESS engine

This section focuses on accessing SAS data sets.

The SQL Procedure

The following table shows SAS terms and the equivalent SQL terminology:

SAS Term	SQL Term
Data Set	Table
Observation	Row
Variable	Column

When using SQL to access a table, remember that you are accessing a SAS data set.

The SQL Procedure: Syntax Overview

The PROC SQL statement signals the start of an SQL procedure.

PROC SQL;

The QUIT statement ends an SQL procedure.

QUIT;

The SQL Procedure: Syntax Overview

An SQL SELECT statement (also called a *query*) is submitted to query SAS tables. A query contains smaller building blocks named *clauses*.

The following clauses are discussed in this section:

- SELECT
- FROM
- WHERE

The SELECT Clause

The SELECT clause identifies columns to include in the query result.

```
SELECT var-1, var-2 ...
```

Columns listed in the SELECT clause are separated by commas. There is no comma following the last column in the list.

```
SELECT *
```

To select all columns, use an asterisk in place of the column names.

The FROM Clause

The FROM clause identifies the SAS table(s) from which to read.

FROM *SAS-table ...*

The WHERE Clause

The WHERE clause identifies a condition that must be satisfied for a row to be included in the PROC SQL output. A WHERE clause can contain any of the columns in a table, including unselected columns.

WHERE *where-expression*

 The *where-expression* can be a compound expression using logical operators.

Using PROC SQL to Query a Table

A query identifies the columns to include in the result, the table to be queried, and a WHERE clause, if desired.

General form of a PROC SQL query:

```
LIBNAME libref 'SAS-data-library';  
PROC SQL;  
    SELECT var-1, var-2...  
        FROM SAS-table-1...  
        <WHERE where-expression  
    ;  
QUIT;
```

A RUN statement is not needed because the query is executed when the semicolon is reached.

The CREATE TABLE Statement

Use the CREATE TABLE statement to create a table with the results of an SQL query.

General form to create a table from a query:

```
LIBNAME libref 'SAS-data-library';  
PROC SQL;  
    CREATE TABLE table-name AS  
    SELECT var-1, var-2...  
        FROM SAS-table-1...  
        <WHERE where-expression>  
;  
QUIT;
```

Using PROC SQL to Join Tables

To join two or more SAS tables, list them in the FROM clause separated by commas.

General form of an SQL inner join:

```
LIBNAME libref 'SAS-data-library';  
PROC SQL;  
    SELECT var-1, var-2...  
        FROM SAS-table-1, SAS-table-2...  
        <WHERE where-expression  
    ;  
QUIT;
```

The WHERE Clause

A WHERE clause is used to specify the join criteria and possibly other subsetting criteria.

WHERE *join-condition(s)*
 <**AND** *other subsetting conditions*>

join-condition can be any valid SAS condition.

When the conditions are met, the rows are displayed in the output.

Advantages of PROC SQL and the DATA Step

With the SQL procedure you can

join tables and produce a report in one step without creating a SAS data set

join tables without presorting the data

use complex matching criteria.

With the DATA step you can

create multiple data sets

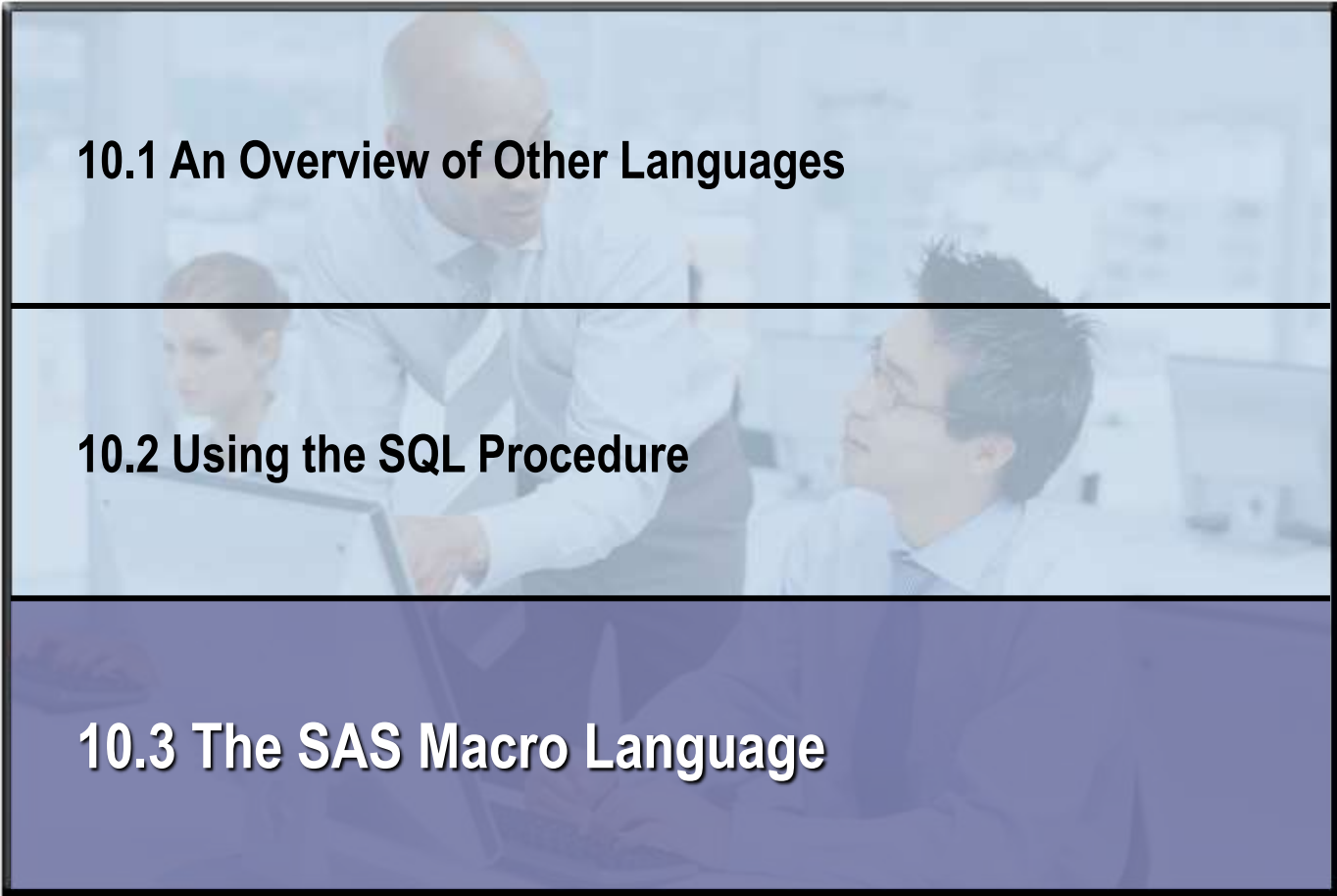
direct output to data sets based on data set contributors

use First. and Last. processing

use DO loops and arrays

perform complex data manipulation.

Chapter 10: Other SAS Languages



10.1 An Overview of Other Languages

10.2 Using the SQL Procedure

10.3 The SAS Macro Language

Objectives

- State the purpose of the macro facility.
- Describe the two types of macro variables.
- Create and use macro variables.
- Display the values of macro variables in the SAS log.

Purpose of the Macro Facility

The *macro facility* is a text-processing facility that supports symbolic substitution within SAS code.

The macro facility enables you to

- create and resolve macro variables anywhere within a SAS program
- write and call macro programs (macros) that generate custom SAS code.

This section focuses on **macro variables**.

Types of Macro Variables

There are two types of macro variables:

- *Automatic* macro variables store system-supplied information such as date, time, or operating system name.
- *User-defined* macro variables store user-supplied information in symbolic names.

System-Defined Automatic Macro Variables

Automatic macro variables are set at SAS invocation and are always available. These include the following:

Name	Description
SYSDATE	Date of SAS invocation (DATE7.)
SYSDATE9	Date of SAS invocation (DATE9.)
SYSDAY	Day of the week of SAS invocation
SYSTIME	Time of SAS invocation
SYSSCP	Abbreviation for the operating system: OS, WIN, HP 64, and so on
SYSVER	Release of SAS software being used.

To refer to a macro variable, use *¯o-variable-name*.

Using Automatic Macro Variables

Automatic macro variables can be used to avoid hardcoding values. In the program below macro variables are referenced to display system information within footnotes.

```
proc print data=orion.customer_type noobs;  
title "Listing of Customer Type Data Set" (3)  
footnote1 "Created &sysptime &sysday, &sysdate9";  
footnote2 "on the &sysrcp System Using Release &sysver";  
run; (4) (5)
```

Macro variable references are replaced with the values of the corresponding macro variables.

Using Automatic Macro Variables

The automatic macro variable references are resolved **prior to compilation.**

```
proc print data=orion.customer type noobs;  
title "Listing of Customer Type Data Set";  
footnote1 "Created 10:24 Wednesday, 25JAN2008";  
footnote2 "on the WIN System Using Release 9.2";  
run;
```

Output: Using Automatic Macro Variables

Partial PROC PRINT Output

Listing of Customer_Type Data Set

Customer_ Type_ID	Customer_Type	Customer_ Group_ID	Customer_Group
1010	Orion Club members inactive	10	Orion Club members
1020	Orion Club members low activity	10	Orion Club members
1030	Orion Club members medium activity	10	Orion Club members
1040	Orion Club members high activity	10	Orion Club members
2010	Orion Club Gold members low activity	20	Orion Club Gold members
2020	Orion Club Gold members medium activity	20	Orion Club Gold members
2030	Orion Club Gold members high activity	20	Orion Club Gold members
3010	Internet/Catalog Customers	30	Internet/Catalog Customers

Created 10:24 Wednesday, 25JAN2008
on the WIN System Using Release 9.2

Use %PUT to Display Macro Variables

The %PUT statement displays the names and values of all automatic macro variables.

```
%put _automatic_;
```

Partial SAS Log

```
1      %put _automatic_;  
AUTOMATIC AFDSID 0  
AUTOMATIC AFDSNAME  
AUTOMATIC AFLIB  
AUTOMATIC AFSTR1  
AUTOMATIC AFSTR2  
AUTOMATIC FSPBDV  
AUTOMATIC SYSBUFR  
AUTOMATIC SYSCC 0  
AUTOMATIC SYSCHARWIDTH 1  
AUTOMATIC SYSCMD  
AUTOMATIC SYSDATE 25JAN08  
AUTOMATIC SYSDATE9 25JAN2008
```

Business Scenario

A developer manually changes the year values each time that the following program is submitted. Use user-defined macro variables to simplify program maintenance.

```
proc print data=orion.order_fact;  
  where year(order_date)=2006;  
  title "Orders for 2006";  
run;  
proc means data=orion.order_fact mean;  
  where year(order_date)=2006;  
  class order_type;  
  var total_retail_price;  
  title "Average Retail Prices for 2006";  
  title2 "by Order_Type";  
run;
```


Creating a Macro Variable

User-defined macro variables can make SAS programs more flexible and easier to modify. Use the %LET macro statement to create a macro variable and assign it a value.

General form of the %LET statement:

```
%LET variable=value;
```

- *variable* follows SAS naming conventions.
- *variable* is created and assigned *value*.
- If *variable* already exists, its *value* is overwritten.

The %LET Statement

value can be any string.

- The length is between 0 (*null value*) and 65,534 (64K) characters.
- A numeric value is stored as a character string.
- Mathematical expressions are **not** evaluated.
- The case of *value* is preserved.
- Quotation marks bounding literals are stored as part of *value*.
- Leading and trailing blanks are **removed** from *value* before the assignment is made.

```
%LET variable=value;
```

Examples of the %LET Statement

```
%let year=2006;  
%let city=Dallas, TX;
```

Name	Value
year	2006
city	Dallas, TX

Poll

Quiz



10.06 Quiz

Complete the rest of the table.

```
%let year=2006;  
%let city=Dallas, TX;  
%let fname=      Marie    ;  
%let name=" Marie Hudson ";  
%let total=10+2;
```

Name	Value
year	2006
city	Dallas, TX

10.06 Quiz – Correct Answer

Complete the rest of the table.

```
%let year=2006;  
%let city=Dallas, TX;  
%let fname=      Marie    ;  
%let name=" Marie Hudson ";  
%let total=10+2;
```

Name	Value
year	2006
city	Dallas, TX
fname	Marie
name	" Marie Hudson "
total	10+2

Create and Use a Macro Variable

Use a %LET statement to create a user-defined macro variable, **year**, and a macro variable reference, **&year**, to obtain the value of the macro variable.

```
%let year=2006;
proc print data=orion.order_fact;
  where year(order_date)= &year;
  title "Orders for &year";
run;
proc means data=orion.order_fact mean;
  where year(order_date)= &year;
  class order_type;
  var total_retail_price;
  title "Average Retail Prices for &year";
  title2 "by Order_Type";
run;
```

Resulting Code: After Symbolic Substitution

The macro variable references are resolved prior to compilation. The references are replaced with the corresponding text value. The resulting code is sent to the compiler.

```
proc print data=orion.order fact;  
  where year(order_date)= 2006;  
  title "Orders for 2006";  
run;  
proc means data=orion.order fact mean;  
  where year(order_date)= 2006;  
  class order_type;  
  var total_retail_price;  
  title "Average Retail Prices for 2006";  
  title2 "by Order_Type";  
run;
```


Poll

Quiz



10.07 Quiz

Examine the program below. What change(s) must be made to generate reports for 2007?

```
%let year=2006;
proc print data=orion.order_fact;
    where year(order_date)= &year;
    title "Orders for &year";
run;
proc means data=orion.order_fact mean;
    where year(order_date)= &year;
    class order_type;
    var total_retail_price;
    title "Average Retail Prices for &year";
    title2 "by Order_Type";
run;
```

10.07 Quiz – Correct Answer

Examine the program below. What change(s) must be made to generate reports for 2007? **Change the value assigned in the %LET statement.**

```
%let year=2007;  
proc print data=orion.order_fact;  
    where year(order_date)= &year;  
    title "Orders for &year";  
run;  
proc means data=orion.order_fact mean;  
    where year(order_date)= &year;  
    class order_type;  
    var total_retail_price;  
    title "Average Retail Prices for &year";  
    title2 "by Order_Type";  
run;
```

Displaying Macro Variable Values in the Log

Enable the SYMBOLGEN system option to write a message to the SAS log each time that a macro variable is resolved.

General form of the SYMBOLGEN system option:

```
OPTIONS SYMBOLGEN;  
OPTIONS NOSYMBOLGEN;
```



The default option is NOSYMBOLGEN.

Displaying Macro Variable Values

Use the SYMBOLGEN option to see the result of macro variable resolution in the SAS log.

```
options symbolgen;  
%let year=2006;  
proc means data=orion.order_fact mean;  
  where year(order_date)=&year;  
  class order_type;  
  var total_retail_price;  
  title "Average Retail Prices for &year";  
  title2 "by Order_Type";  
run;
```

Displaying Macro Variable Values

A message is written to the SAS log each time that a macro variable is resolved.

```
59  options symbolgen;
60  %let year=2006;
61  proc means data=orion.order_fact mean;
62      where year(order_date)=&year;
SYMBOLGEN:  Macro variable YEAR resolves to 2006
63      class order_type;
64      var total_retail_price;
SYMBOLGEN:  Macro variable YEAR resolves to 2006
65      title "Average Retail Prices for &year Orders";
66      title2 "by Order_Type";
67  run;
```

Using Macro Variables within a SAS Literal

The resulting code might require quotation marks. If so, the macro variable reference should be enclosed in double quotation marks.

```
%let site=Melbourne;  
proc print data=orion.employee_addresses;  
  where City="&site";  
  var Employee_ID Employee_Name;  
  title 'Employees from &site';  
run;
```

Notice that the macro variable reference, **&site**, is enclosed in double quotation marks in the first reference above and single quotation marks in the second reference.

Substitution within a SAS Literal

Double quotation marks allow macro variable resolution and single quotation marks prevent macro variable resolution.

```
12  %let site=Melbourne;
13  proc print data=orion.employee_addresses;
14      where City="&site";
15      var Employee_ID Employee_Name;
16      title 'Employees from &site';
17  run;
```

NOTE: There were 41 observations read from the data set
ORION.EMPLOYEE_ADDRESSES.

WHERE City='Melbourne';

Site resolved in double quotation marks.

Partial PROC PRINT Output

Obs	Employee_ID	Employee_Name
2	120145	Aisbitt, Sandy
24	120168	Barcoe, Selina

Site did not resolve in single quotation marks.

Substitution within a SAS Literal

There is no SYMBOLGEN message for the second reference to **site** because it is not resolved.

```
68  options symbolgen;
69  %let site=Melbourne;
70  proc print data=orion.employee_addresses;
71      where City="&site";
SYMBOLGEN:  Macro variable SITE resolves to Melbourne
72      var Employee_ID Employee_Name;
73      title 'Employees from &site';
74  run;
```

NOTE: There were 41 observations read from the data set
ORION.EMPLOYEE_ADDRESSES.
WHERE City='Melbourne';

Use %PUT to Display Macro Variables

To display the names and values of all user-defined macro variables, use this form of the %PUT statement:

```
%put _user_;
```

Partial SAS Log

```
136 %put _user_;  
GLOBAL SITE Melbourne  
GLOBAL YEAR 2007
```

To display both user-defined and automatic macro variables, use this form of the %PUT statement:

```
%put _all_;
```

Bonus: Graphics Techniques



PROC SGPLOT

PROC SGPANEL

Introduction to SAS ODS Graphics

ODS Statistical Graphics (also known as ODS Graphics) is functionality for easily creating statistical graphics. It is available in a number of SAS products, including SAS/STAT, SAS/ETS, SAS/QC, and SAS/GRAPH software.

Documentation (1652 pages):

SAS® 9.4 ODS Graphics: Procedures Guide, Sixth Edition

<https://support.sas.com/documentation/cdl/en/grstatproc/69716/PDF/default/grstatproc.pdf>

SAS ODS Graphics Procedures

There are five ODS Graphics procedures. Each has a specific purpose:

- **SGPLOT**

- creates single-cell plots with a variety of plot and chart types and overlays.

- **SGPANEL**

- creates classification panels for one or more classification variables. Each graph cell in the panel can contain either a simple plot or multiple, overlaid plots.

Note: The SGPLOT and SGPANEL procedures largely support the same types of plots and charts and have an almost identical syntax. The main distinction between the two procedures is that the SGPANEL procedure produces a panel of graphs, one for each level of a classification variable.

SAS ODS Graphics Procedures (Self-Study)

- SGSCATTER

- creates scatter plot panels and scatter plot matrices with optional fits and ellipses.

- SGRENDER

- produces graphs from graph templates that are written in the Graph Template Language. You can also render a graph from a SAS ODS Graphics Editor (SGE) file.

- SGDESIGN

- creates graphical output based on a graph file that has been created by using the ODS Graphics Designer application.

List of Plots and Charts

Band plot	Line chart	Text Inset
Bar chart	Line, drop	Text plot
Block plot	Line, parameterized	Vector plot
Box plot	Line, reference	Waterfall chart
Bubble plot	Loess plot	
Density plot	Needle plot	
Dot plot	Penalized B-Spline plot	
Ellipse plot	Regression plot	
Fringe plot	Scatter plot	
Heat map	Series plot	
High-Low plot	Spline plot	
Histogram	Step plot	

Producing Charts with the SGPLOT Procedure

General form of the PROC SGPLOT procedure:

```
PROC SGPLOT DATA=SAS-data-set;  
    TYPE1 chart-variable(s) . . . </ options>;  
    TYPE2 chart-variable(s) . . . </ options>;  
RUN;
```

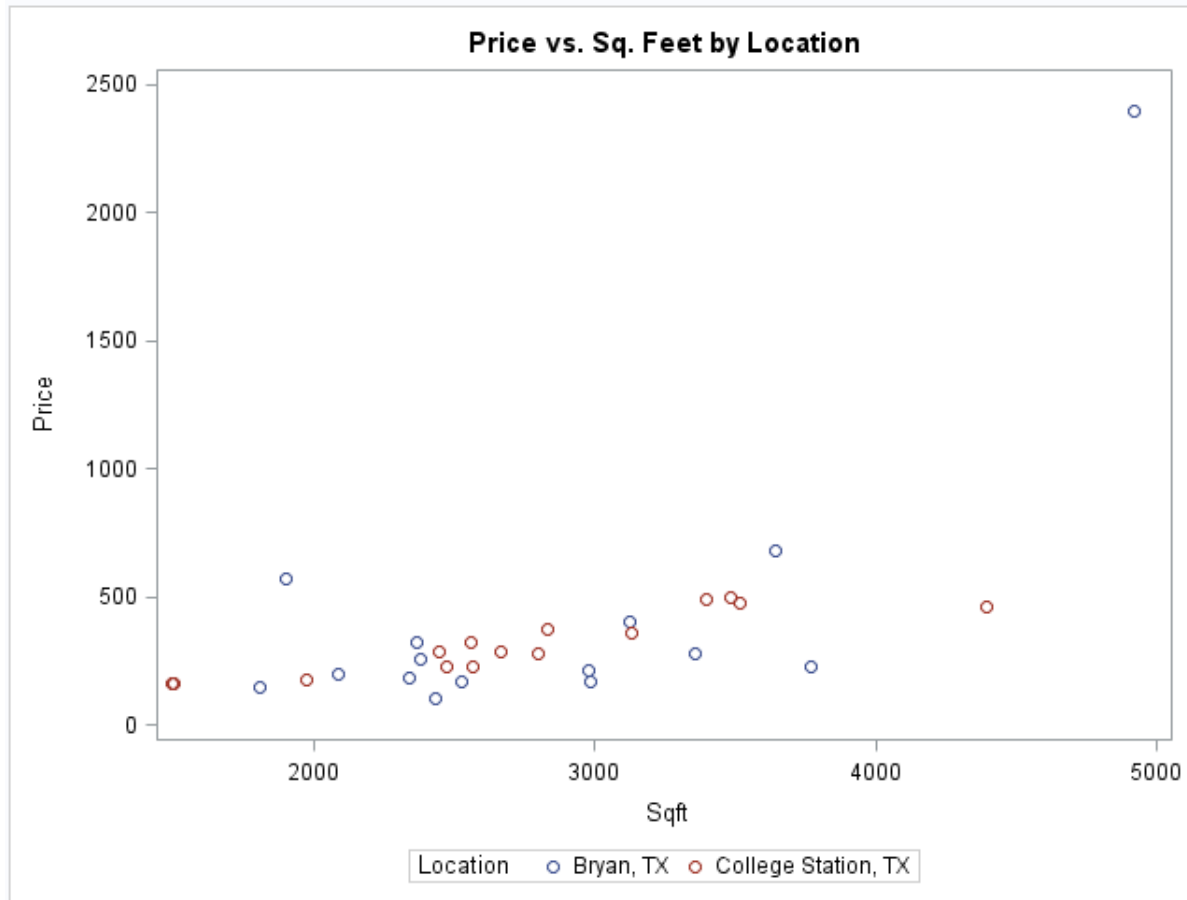

Producing Charts with the SGPLOT Procedure

Example:

```
title 'Price vs. Sq. Feet by Location';  
proc sgplot data=bcs;  
    scatter x=sqft y=price /group=location;  
run;
```

Producing Charts with the SGPLOT Procedure

Results:



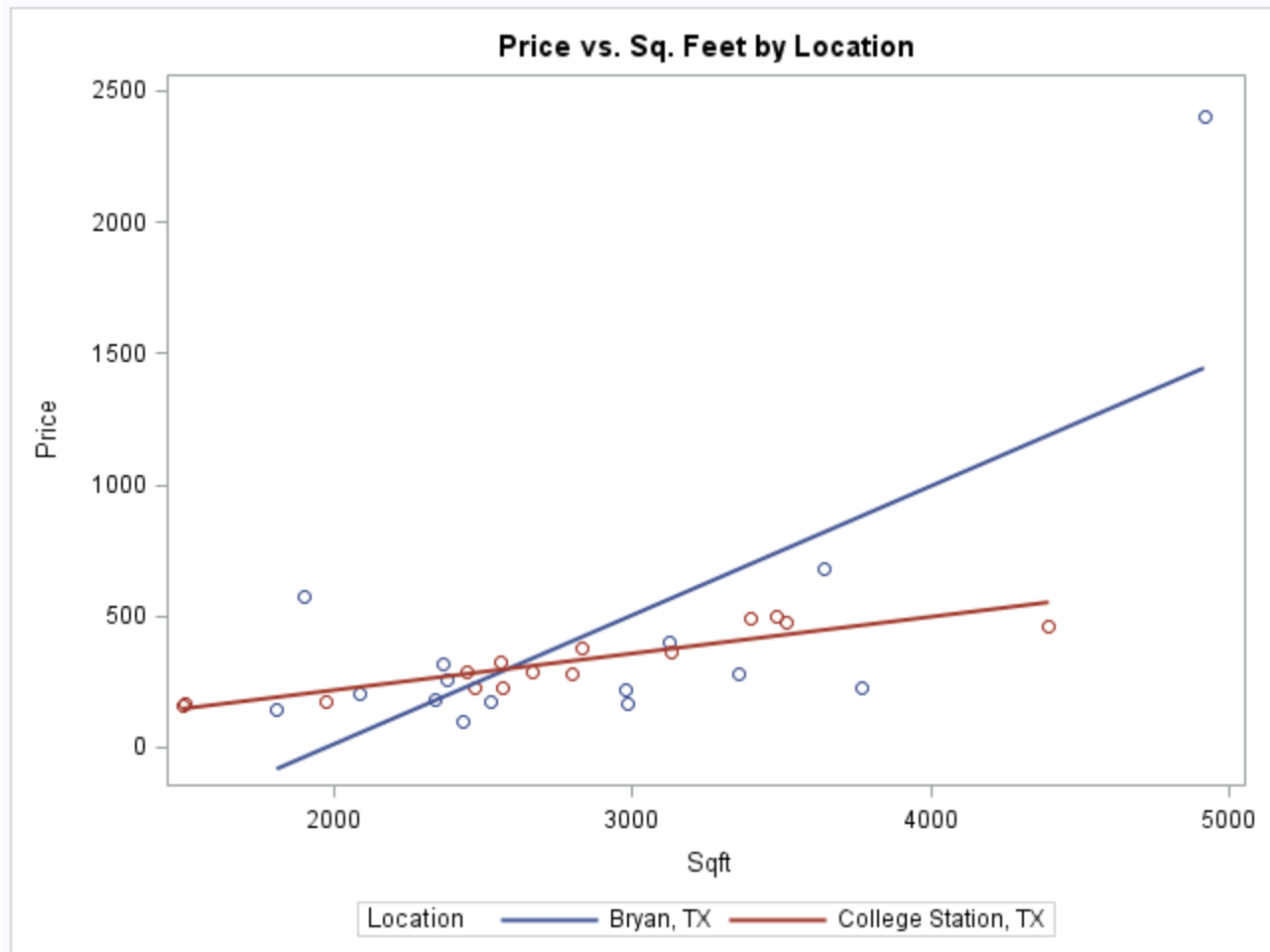
Fit Lines with the SGPLOT Procedure

Example:

```
title 'Price vs. Sq. Feet by Location';  
proc sgplot data=bcs;  
    reg x=sqft y=price /group=location;  
run;
```

Fit Lines with the SGPLOT Procedure

Results:



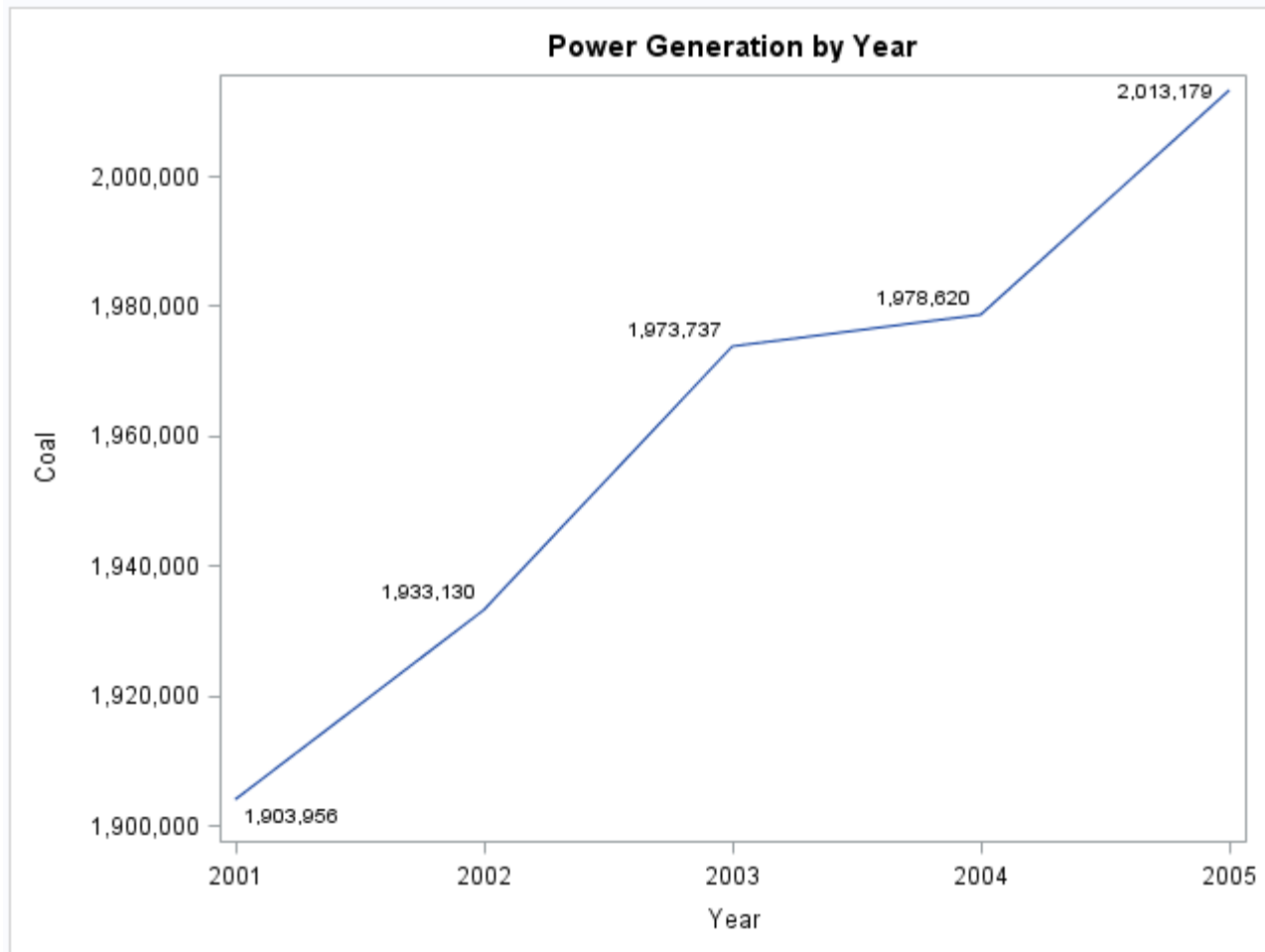
Producing Lines with the SGPLOT Procedure

Example:

```
title 'Power Generation by Year';  
proc sgplot data=sashelp.electric;  
    where year >= 2001  
        and customer="Residential";  
    series x=year y=coal / datalabel;  
run;
```

Producing Lines with the SGPLOT Procedure

Results:



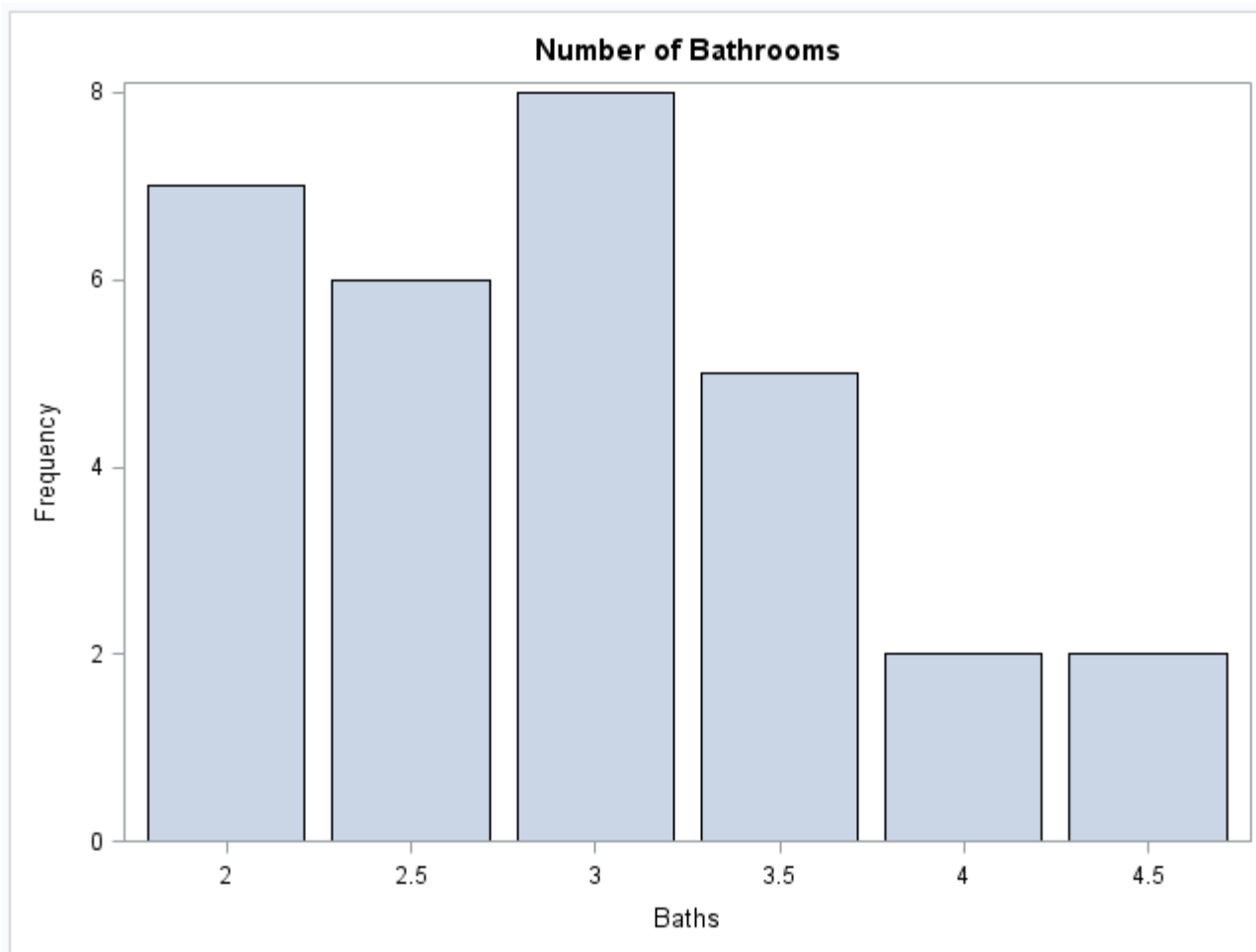
Bar Plots with the SGPLOT Procedure

Example:

```
title 'Number of Bathrooms';  
proc sgplot data=bcs;  
    vbar baths;  
run;
```

Bar Plots with the SGPLOT Procedure

Results:



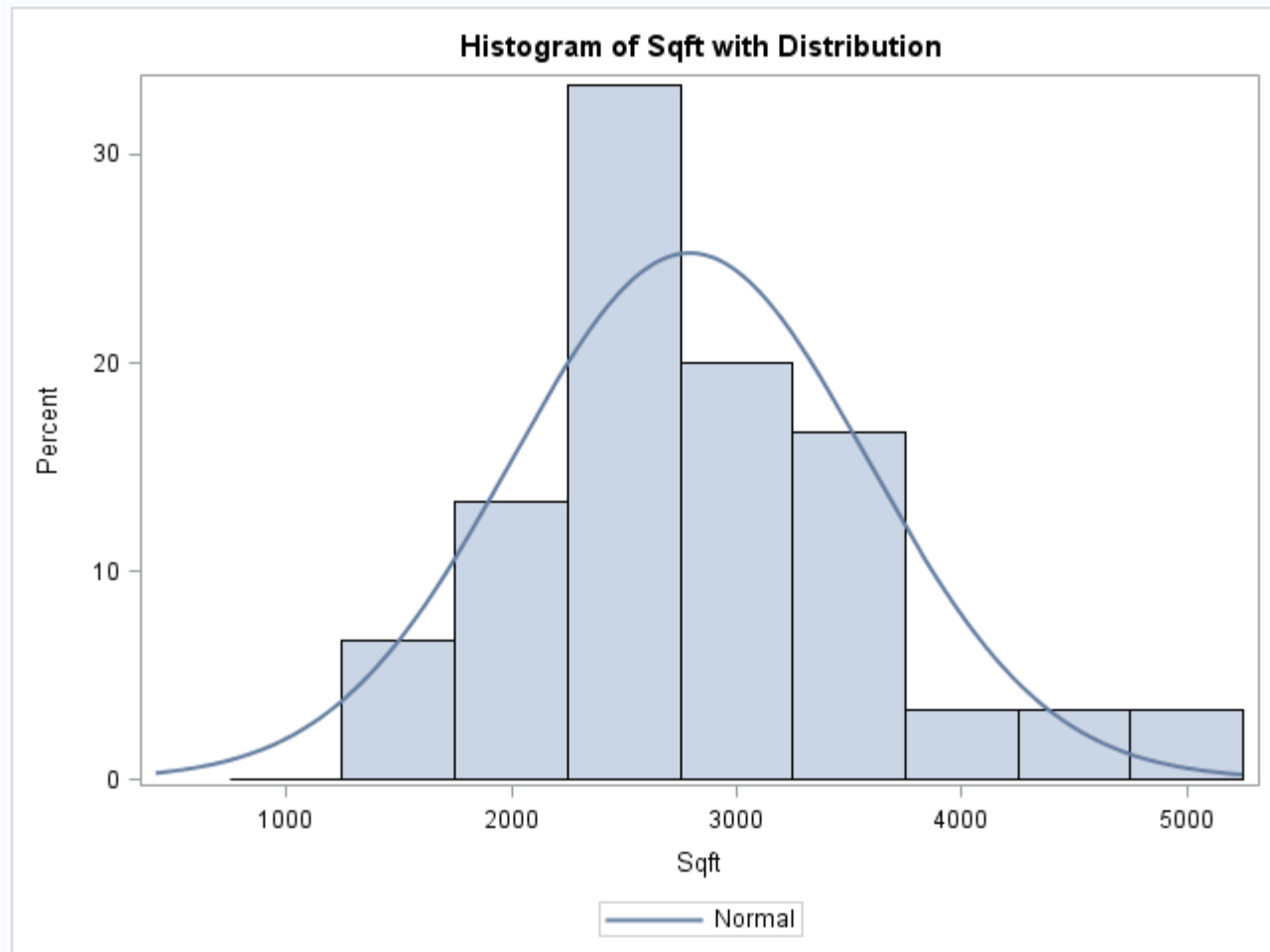
Histograms with the SGPLOT Procedure

Example:

```
title 'Histogram of Sqft with Distribution';  
proc sgplot data=bcs;  
    histogram sqft / binwidth=500 ;  
    density sqft;  
run;
```

Histograms with the SGPLOT Procedure

Results:



Boxplots with the SGPLOT Procedure

Example:

```
title 'Real Estate Sales Prices';  
proc sgplot data=bcs;  
    vbox price /group=location;  
run;
```

Boxplots with the SGPLOT Procedure

Results:



Producing Charts with the SGPANEL Procedure

General form of the PROC SGPANEL procedure:

```
PROC SGPANEL DATA=SAS-data-set;  
    PANELBY group-variable;  
        TYPE1 chart-variable(s) . . . </ options>;  
        TYPE2 chart-variable(s) . . . </ options>;  
RUN;
```

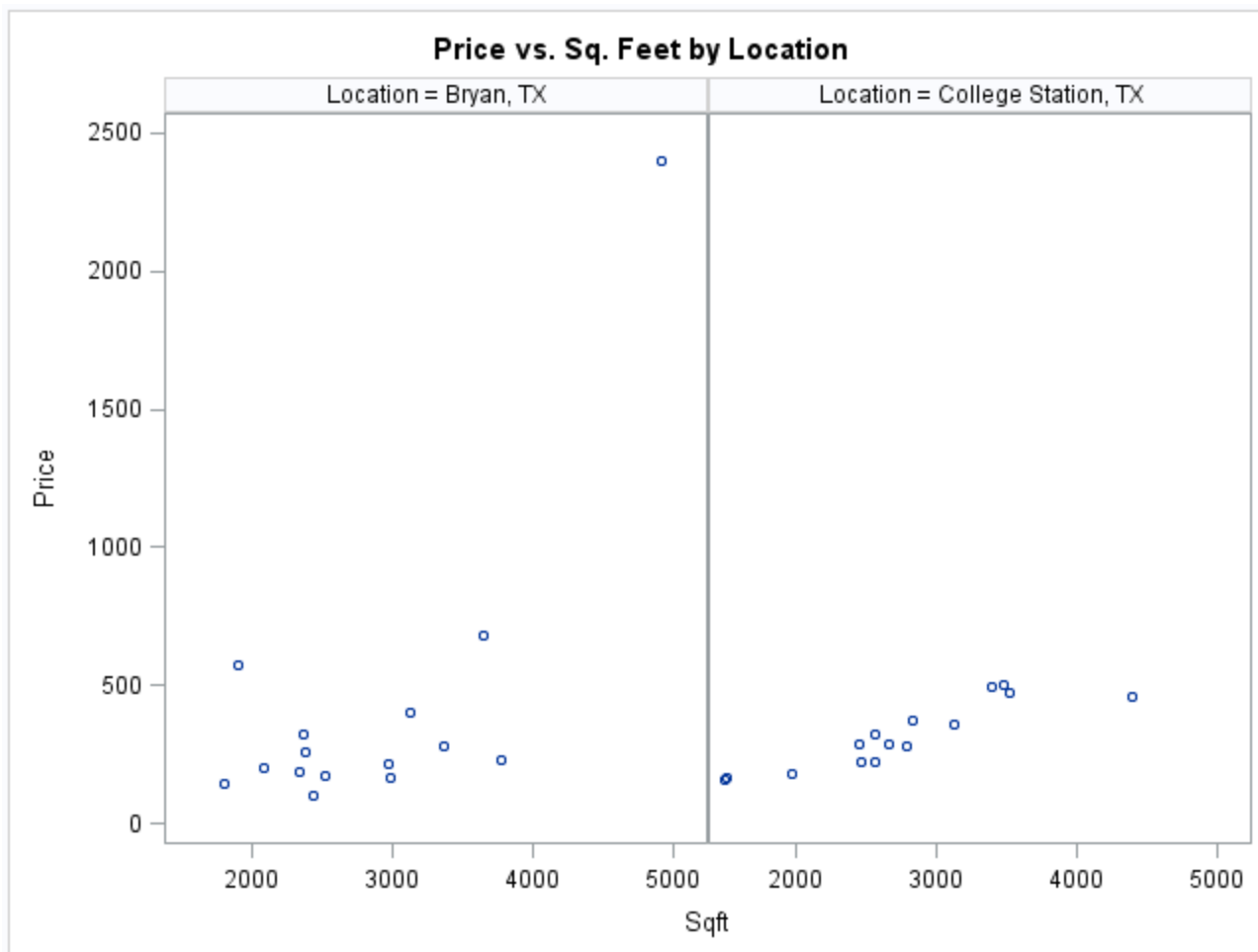
Producing Charts with the SG PANEL Procedure

Example:

```
title 'Price vs. Sq. Feet by Location';  
proc sgpanel data=bcs;  
    panelby location;  
    scatter x=sqft y=price;  
run;
```

Producing Charts with the SGPanel Procedure

Results:



STAT 604 Takeaways

1. Know thy data
2. Pay attention to detail
3. No errors in log does not mean results are correct.
4. Think!

