

SAS® Programming for R Users

Course Notes

SAS® Programming for R Users Course Notes was developed by Jordan Bakerman. Additional contributions were made by Jay Laramore, Stephen Mistler, Danny Modlin, Kathy Passarella, Mike Patetta, Andy Ravenna, and Cat Truxillo. Editing and production support was provided by the Curriculum Development and Support Department.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

SAS® Programming for R Users Course Notes

Copyright © 2016 SAS Institute Inc. Cary, NC, USA. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

Book code E70576, course code LWSP4R/SP4R, prepared date 22Mar2016.

LWSP4R_001

ISBN 978-1-62960-200-4

Table of Contents

Course Description	viii
Prerequisites	ix
Chapter 1 Introduction.....	1-1
1.1 Introduction.....	1-3
Demonstration: Navigating SAS Studio	1-8
Demonstration: Writing a Program in SAS Studio	1-14
Demonstration: Using Tasks	1-21
Demonstration: Using Snippets.....	1-26
Exercises.....	1-31
Demonstration: Bayesian Logistic Regression	1-32
Demonstration: Poker Simulation.....	1-40
Demonstration: Determining Power Using Simulation	1-43
Demonstration: Calling R from SAS	1-46
Demonstration: Navigating the Online Documentation.....	1-51
1.2 SAS Programming (Self-Study)	1-58
1.3 Accessing Data in SAS Libraries.....	1-64
Demonstration: Creating the Course Data and Library.....	1-71
1.4 Solutions	1-72
Solutions to Student Activities (Polls/Quizzes).....	1-72
Chapter 2 Importing and Reporting Data.....	2-1
2.1 The DATA Step and Manual Data Entry.....	2-3
Demonstration: Creating and Viewing the Data Table.....	2-6
2.2 Importing Data.....	2-9
Demonstration: Importing Data Using a DATA Step.....	2-13
Demonstration: Importing Data Using PROC IMPORT.....	2-14
Demonstration: Creating a SAS Data Set from Delimited Data by Hand.....	2-16
2.3 Reporting the Data	2-17
2.4 Enhanced Reporting.....	2-23

Demonstration: Labels, Formats, and Informats	2-29
Exercises.....	2-31
2.5 Solutions	2-33
Solutions to Exercises	2-33
Solutions to Student Activities (Polls/Quizzes).....	2-38
Chapter 3 Creating New Variables, Functions, and Data Tables	3-1
3.1 Creating New Variables	3-3
3.2 Creating and Using Functions.....	3-13
Demonstration: Navigating to the SAS Online Functions Documentation.....	3-22
3.3 Subsetting and Concatenating Data Tables	3-29
Exercises.....	3-39
3.4 Solutions	3-42
Solutions to Exercises	3-42
Solutions to Student Activities (Polls/Quizzes).....	3-46
Chapter 4 Random Number Generation and Plotting	4-1
4.1 DO Loop and Random Number Generation	4-3
Demonstration: DO Loop and Random Number Generation.....	4-13
4.2 Single-Cell Plotting.....	4-16
Demonstration: Creating Standard R Plots	4-25
Demonstration: Enhancing the Plot	4-35
4.3 Multi-Cell Plotting.....	4-38
Demonstration: Multi-Cell Plotting	4-45
Exercises.....	4-48
4.4 Solutions	4-55
Solutions to Exercises	4-55
Solutions to Student Activities (Polls/Quizzes).....	4-66
Chapter 5 Descriptive Procedures, Output Delivery System, and Macros.....	5-1
5.1 CORR, FREQ, MEANS, and UNIVARIATE Procedures	5-3
Demonstration: Ames Home Sales Data Exploration	5-15

5.2	Output Delivery System (ODS)	5-20
	Demonstration: Selecting Output with ODS TRACE	5-24
	Demonstration: ODS Tables and Graphics	5-26
	Demonstration: Creating Data Tables with ODS	5-30
5.3	Creating Macro Variables.....	5-32
	Demonstration: Creating and Using Macro Variables.....	5-42
5.4	Creating Macro Programs	5-45
	Demonstration: A Macro Program to Generate Data, Summary Statistics, and Plots	5-55
	Demonstration: A Macro Program for Iterative Processing	5-58
	Exercises.....	5-60
5.5	Solutions	5-67
	Solutions to Exercises	5-67
	Solutions to Student Activities (Polls/Quizzes).....	5-81
Chapter 6	Analyzing the Data via Inferential Procedures.....	6-1
6.1	Linear Models	6-3
	Demonstration: Multiple Linear Regression.....	6-10
	Demonstration: Polynomial Regression.....	6-15
	Demonstration: Two-Way ANOVA.....	6-27
	Demonstration: ANCOVA	6-32
	Demonstration: Stepwise Selection with PROC GLMSELECT	6-43
	Demonstration: Polynomial Regression with PROC GLMSELECT	6-48
6.2	Generalized Linear Models.....	6-52
	Demonstration: Logistic Regression.....	6-59
	Demonstration: GENMOD Procedure	6-69
6.3	Mixed Models	6-78
	Demonstration: Two-Way Mixed Model	6-84
6.4	Other Procedures.....	6-90
	Exercises.....	6-99
6.5	Solutions	6-116
	Solutions to Exercises	6-116
	Solutions to Student Activities (Polls/Quizzes).....	6-154

Chapter 7 Interactive Matrix Language (IML)	7-1
7.1 The Basics (Self-Study)	7-3
Demonstration: Working in Interactive Mode (SAS Studio)	7-11
Demonstration: Basic Matrix Operations.....	7-12
Exercises.....	7-15
7.2 Modules and Subroutines.....	7-17
Demonstration: Simple Linear Regression	7-23
Demonstration: Navigating the SAS/IML Documentation	7-30
Demonstration: Creating Functions and Subroutines.....	7-43
7.3 Calling SAS Data Sets and Procedures.....	7-45
Demonstration: Calling SAS Data Sets from SAS/IML	7-52
Demonstration: Submitting SAS Procedures	7-59
7.4 Simulations	7-61
Demonstration: The Monty Hall Problem.....	7-68
Demonstration: Sampling Distribution – Method 1	7-72
Demonstration: Sampling Distribution – Method 2	7-74
Demonstration: Sampling Distribution – Method 3	7-77
Exercises.....	7-79
7.5 Solutions	7-86
Solutions to Exercises	7-86
Solutions to Student Activities (Polls/Quizzes).....	7-99
Chapter 8 A Bridge between SAS and R	8-1
8.1 Calling R from IML	8-3
Demonstration: Bootstrap with R from IML	8-12
Exercises.....	8-17
8.2 Calling R from Base SAS Java API (Self-Study)	8-18
8.3 Calling R from SAS Enterprise Miner (Self-Study)	8-26
Demonstration: Getting Started with Enterprise Miner and the Open Source Integration Node.....	8-37
8.4 Solutions	8-54
Solutions to Exercises	8-54

Solutions to Student Activities (Polls/Quizzes).....	8-56
--	------

Course Description

This course is designed for experienced R users who want to transfer their programming skills to SAS. Emphasis will be placed on programming and not statistical theory or interpretation. Students of this course should have knowledge of plotting, manipulating data, iterative processing, creating functions, applying functions, linear models, generalized linear models, mixed models, stepwise model selection, matrix algebra, and statistical simulations.

To learn more...



For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the web at <http://support.sas.com/training/> as well as in the Training Course Catalog.



For a list of other SAS books that relate to the topics covered in this course notes, USA customers can contact the SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the SAS Bookstore on the web at <http://support.sas.com/publishing/> for a complete list of books and a convenient order form.

Prerequisites

There are no prerequisites for this course.

x

For Your Information

Chapter 1 Introduction

1.1 Introduction.....	1-3
Demonstration: Navigating SAS Studio	1-8
Demonstration: Writing a Program in SAS Studio	1-14
Demonstration: Using Tasks	1-21
Demonstration: Using Snippets	1-26
Exercises	1-31
Demonstration: Bayesian Logistic Regression	1-32
Demonstration: Poker Simulation	1-40
Demonstration: Determining Power Using Simulation.....	1-43
Demonstration: Calling R from SAS.....	1-46
Demonstration: Navigating the Online Documentation	1-51
1.2 SAS Programming (Self-Study)	1-58
1.3 Accessing Data in SAS Libraries.....	1-64
Demonstration: Creating the Course Data and Library.....	1-71
1.4 Solutions	1-72
Solutions to Student Activities (Polls/Quizzes)	1-72

1.1 Introduction

Objectives

- Define SAS.
- Discuss the three SAS interfaces.
- Explore SAS Studio.

3

What Is SAS?

SAS is a suite of business solutions and technologies to help organizations solve business problems.



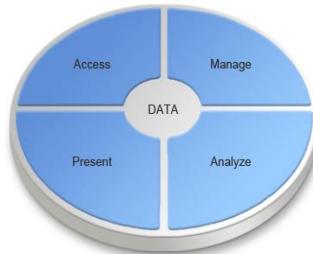
4

SAS is powered by high-performance analytics, which are thoroughly tested before coming to market.

Functionality of SAS

SAS enables you to do the following tasks:

- access and manage data across multiple sources
- perform analyses and deliver information across your organization



5

The functionality of SAS is built around these four data-driven tasks that are common to virtually any application:

- data access
- data management
- data analysis
- data presentation

SAS Interfaces

Since its inception, SAS software evolved significantly with the changes in computer technology. This evolution resulted in three unique SAS interfaces:

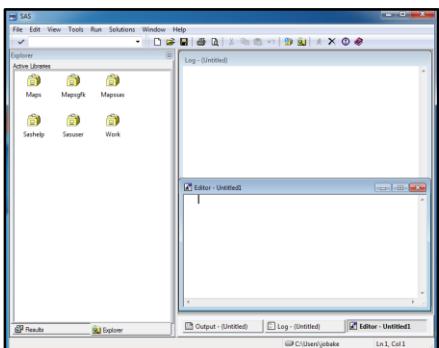
- SAS windowing environment
- SAS Enterprise Guide
- SAS Studio



6

SAS Interfaces

Since its inception, SAS software evolved significantly with the changes in computer technology. This evolution resulted in three unique SAS interfaces: **SAS windowing environment**, SAS Enterprise Guide, SAS Studio.



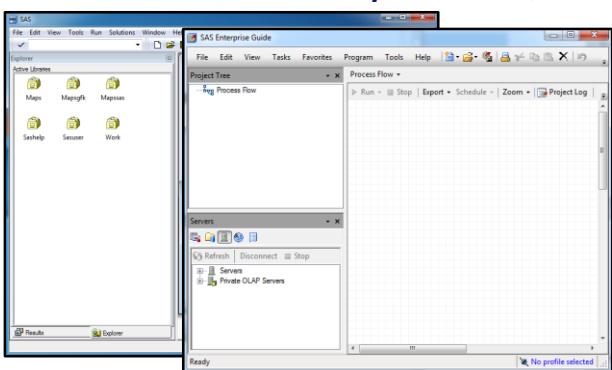
7

...

The SAS windowing environment is the original interface that is used to access, manage, analyze, and report data. As an experienced programmer, the windowing environment might feel the most natural because it is the most basic interface of SAS. It provides an Editor window in which you can write and submit code without the use of any point-and-click features.

SAS Interfaces

Since its inception, SAS software evolved significantly with the changes in computer technology. This evolution resulted in three unique SAS interfaces: SAS windowing environment, **SAS Enterprise Guide**, and SAS Studio.



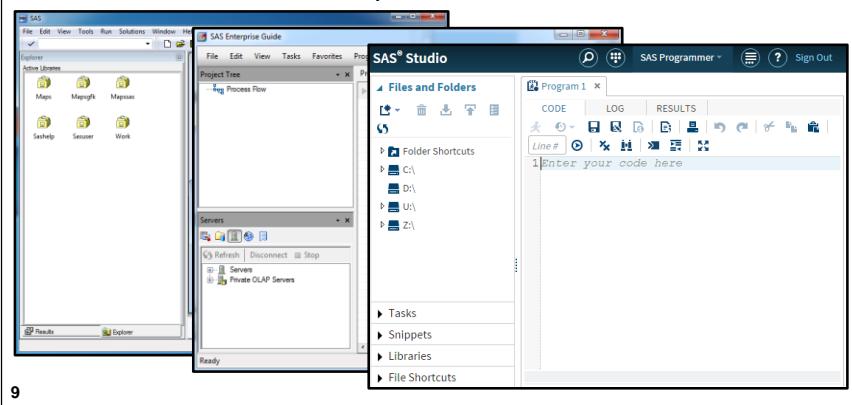
8

...

SAS Enterprise Guide is configured to access SAS on a local or remote server. SAS Enterprise Guide has point-and-click wizards and tasks for SAS procedures and a robust programming interface.

SAS Interfaces

Since its inception, SAS software evolved significantly with the changes in computer technology. This evolution resulted in three unique SAS interfaces: SAS windowing environment, SAS Enterprise Guide, and **SAS Studio**.



SAS Studio is the newest interface. It is a web-based interface to SAS that you can use on any computer. It combines functionality from both the windowing environment and Enterprise Guide.

No matter which SAS interface you use, the SAS programming is the same. In addition, they all offer these same basic programming tools:

- an Editor window where you write and submit SAS code
- a log where you view messages from SAS
- a page to view your results

What Is SAS Studio?

- **Consistent** – You learn one interface that you can use throughout your career, as a student, an individual SAS user or consultant, a departmental user, and an enterprise user.
- **Available** – You can use the same interface wherever you need it (a Mac in a dorm, a Windows desktop at work, a laptop at home, and an iPad on the road).
- **Assistive** – For programmers, the code is front and center, but you can use point-and-click functions such as code-generating tasks or process flows to help, if you need them.

How Does SAS Studio Work?

- SAS Studio is accessed from your browser.
- When you run a program, SAS Studio connects to SAS in order to execute the SAS code.
- SAS processes the code and the results are returned to SAS Studio.

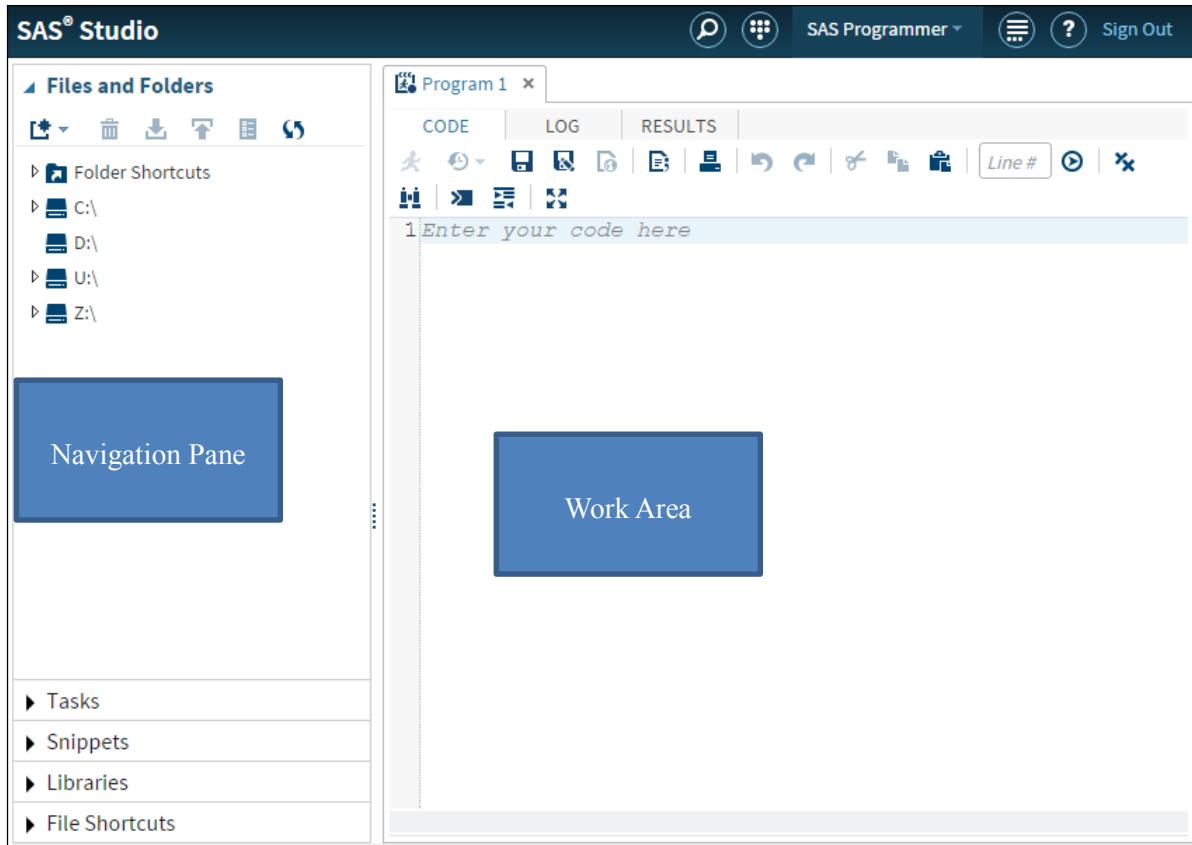


11



Navigating SAS Studio

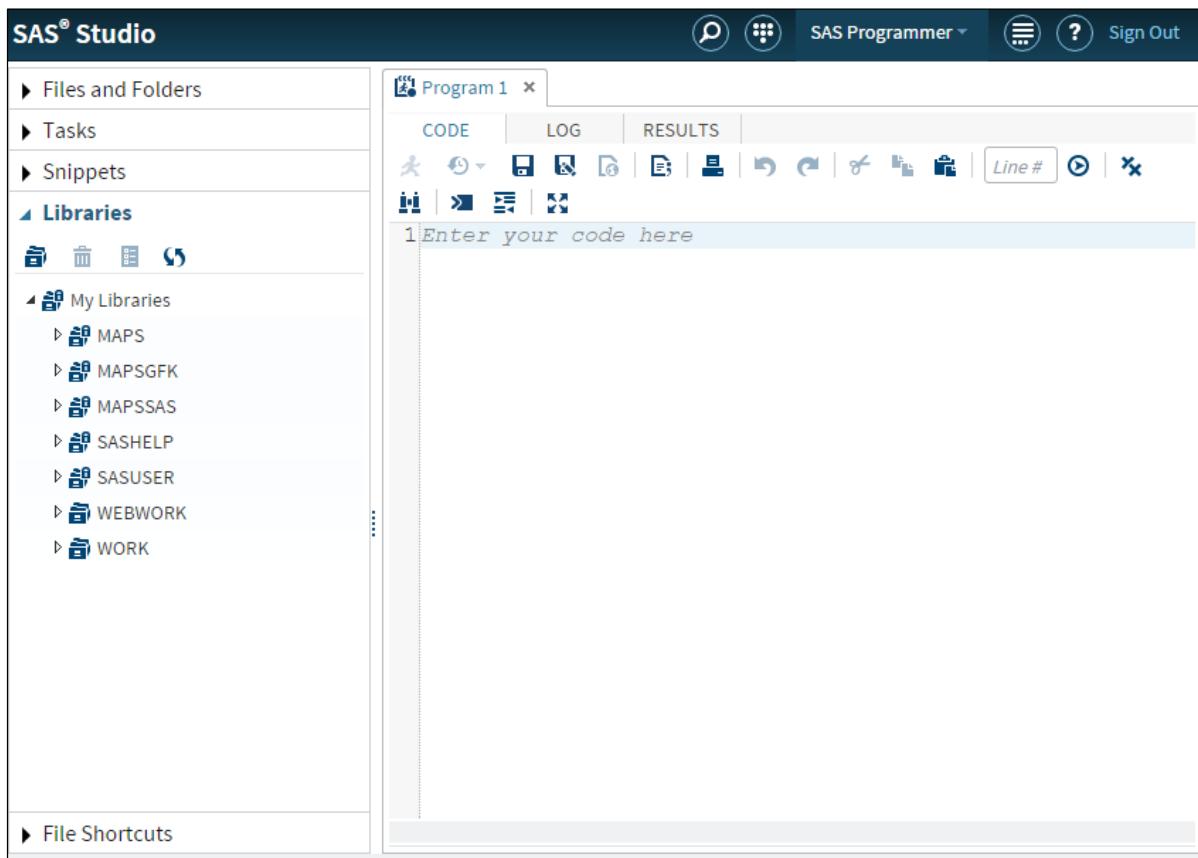
1. Open your SAS Studio session to view the interface.



The SAS Studio interface is separated into the Navigation pane on the left and the Work area on the right. The Work area displays your programs with tabs for Code, Log, and Results. The Navigation pane provides easy access to your folders and libraries that contain your permanent and temporary data sets.

The Files and Folders tab is displayed by default and gives the user quick access to load data sets and SAS programs.

2. Click the **Libraries** tab in the Navigation pane and select **My Libraries**.



The libraries **MAPS** through **WEBWORK** are permanent libraries. The data displayed in each library is a permanent data set, which users can use at their convenience. The **Work** library is a temporary library. Any data saved to the **Work** library by the user is deleted when the user closes the SAS session. In a later demonstration, you see how to save a new data set to the **Work** library and create a new permanent library. A new permanent library enables the user to load external data a single time and update or use the data table each new session. This heavily reduces the load time and cleaning time of your data because it is done only once.

3. Open the **Sashelp** library and navigate to the **cars** data set. Double-click the data set to open it in the Table Viewer in the Work area.

The screenshot shows the SAS Studio interface. On the left, the Libraries panel is open, displaying various datasets like BMIIMEN, BMT, BURROWS, BUY, BWEIGHT, and CARS, with CARS selected. The main workspace shows the Table Viewer for the SASHHELP.CARS dataset. The viewer has two panes: a left pane for columns and a right pane for data rows. The columns pane lists 15 columns: Make, Model, Type, Origin, DriveTrain, MSRP, Invoice, EngineSize, Cylinders, Horsepower, and three unlabeled columns. The data pane shows 17 rows of car information, including models like Acura MDX, RSX Type S 2dr, TSX 4dr, TL 4dr, 3.5 RL 4dr, 3.5 RL w/Navigation 4dr, NSX coupe 2dr manual S, Audi A4 1.8T 4dr, A41.8T convertible 2dr, A4 3.0 4dr, A4 3.0 Quattro 4dr manual, A4 3.0 Quattro 4dr auto, A6 3.0 4dr, A6 3.0 Quattro 4dr, A4 3.0 convertible 2dr, A4 3.0 Quattro convertible 2dr, and A6 2.7 Turbo Quattro 4dr.

	Make	Model
1	Acura	MDX
2	Acura	RSX Type S 2dr
3	Acura	TSX 4dr
4	Acura	TL 4dr
5	Acura	3.5 RL 4dr
6	Acura	3.5 RL w/Navigation 4dr
7	Acura	NSX coupe 2dr manual S
8	Audi	A4 1.8T 4dr
9	Audi	A41.8T convertible 2dr
10	Audi	A4 3.0 4dr
11	Audi	A4 3.0 Quattro 4dr manual
12	Audi	A4 3.0 Quattro 4dr auto
13	Audi	A6 3.0 4dr
14	Audi	A6 3.0 Quattro 4dr
15	Audi	A4 3.0 convertible 2dr
16	Audi	A4 3.0 Quattro convertible 2dr
17	Audi	A6 2.7 Turbo Quattro 4dr

The **cars** data set contains 428 total rows of data and 15 columns or variables.

You can use the arrows to navigate between pages or the scroll bar at the bottom of the data table to change your view of the data.

In the Columns area of the Table Viewer, notice that all columns are selected by default. Simply clear the check box from a column to remove the column from the viewer.

4. Clear the **Select all** check box and then select **Make**, **Model**, **Type**, **Origin**, **MSRP**, and **Invoice**.

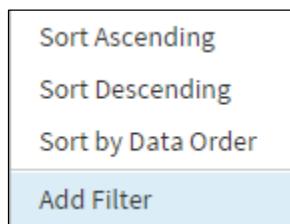
The screenshot shows the SAS Studio interface. On the left, the 'Libraries' pane is open, displaying a list of datasets. The 'CARS' dataset is selected and highlighted with a blue background. In the center, the 'SASHELP.CARS' dataset is displayed in a grid format. The 'Columns' pane on the left lists the columns of the dataset, with checkboxes indicating which columns are selected. The selected columns are: Make, Model, Type, Origin, MSRP, and Invoice. The main grid shows 15 columns of data, with the first two columns being Make and Model. The data includes various car models from brands like Acura, Audi, and Toyota, along with their respective details such as MSRP and origin.

	Make	Model
1	Acura	MDX
2	Acura	RSX Type S 2dr
3	Acura	TSX 4dr
4	Acura	TL 4dr
5	Acura	3.5 RL 4dr
6	Acura	3.5 RL w/Navigation 4dr
7	Acura	NSX coupe 2dr manual S
8	Audi	A4 1.8T 4dr
9	Audi	A41.8T convertible 2dr
10	Audi	A4 3.0 4dr
11	Audi	A4 3.0 Quattro 4dr manual
12	Audi	A4 3.0 Quattro 4dr auto
13	Audi	A6 3.0 4dr
14	Audi	A6 3.0 Quattro 4dr
15	Audi	A4 3.0 convertible 2dr
16	Audi	A4 3.0 Quattro convertible 2dr
17	Audi	A6 2.7 Turbo Quattro 4dr
18	Audi	A6 4.2 Quattro 4dr

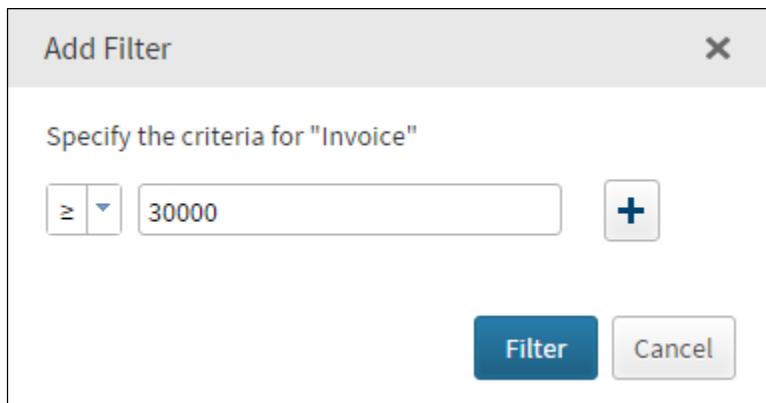
5. To customize the view of the data table, select the arrow next to **Columns** to hide the columns area and then select the **Maximize View** icon.

	Make	Model	Type	Origin	MSRP	Invoice
1	Acura	MDX	SUV	Asia	\$36,945	\$33,337
2	Acura	RSX Type S 2dr	Sedan	Asia	\$23,820	\$21,761
3	Acura	TSX 4dr	Sedan	Asia	\$26,990	\$24,647
4	Acura	TL 4dr	Sedan	Asia	\$33,195	\$30,299
5	Acura	3.5 RL 4dr	Sedan	Asia	\$43,755	\$39,014
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	\$46,100	\$41,100
7	Acura	NSX coupe 2dr manual S	Sports	Asia	\$89,765	\$79,978
8	Audi	A4 1.8T 4dr	Sedan	Europe	\$25,940	\$23,508
9	Audi	A41.8T convertible 2dr	Sedan	Europe	\$35,940	\$32,506
10	Audi	A4 3.0 4dr	Sedan	Europe	\$31,840	\$28,846
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	\$33,430	\$30,366
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	\$34,480	\$31,388
13	Audi	A6 3.0 4dr	Sedan	Europe	\$36,640	\$33,129
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	\$39,640	\$35,992
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	\$42,490	\$38,325
16	Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	\$44,240	\$40,075
17	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	\$42,840	\$38,840
18	Audi	A6 4.2 Quattro 4dr	Sedan	Europe	\$49,690	\$44,936

6. You can right-click a column heading to filter and sort the data table by that column. Right-click the **Invoice** column and select **Add Filter**. Notice that the other options are Sort Ascending, Sort Descending, and Sort by Data Order.



7. Add a filter to select only the rows with Invoice values greater than or equal to \$30,000. Use the drop-down menu to change the filter in the Add Filter window. Add the filter value in the text box. Then click **Filter**.



	Make	Model	Type	Origin	MSRP	Invoice
1	Nissan	Quest SE	Sedan	Asia	\$32,780	\$30,019
2	Mercedes-Benz	C240 4dr	Sedan	Europe	\$32,280	\$30,071
3	BMW	325xi Sport	Wagon	Europe	\$32,845	\$30,110

At the top of the table, you see that the number of filtered rows is 160.

8. As you select options and customize the table, SAS Studio generates SAS code that you can use. To view the query code, click the **Display Query** button on the toolbar.

```

PROC SQL;
  CREATE TABLE work.query AS
    SELECT Make , Model , 'Type'n , Origin , MSRP , Invoice
      FROM sashelp.cars WHERE Invoice >=30000 ORDER BY Invoice;
RUN;
QUIT;

PROC DATASETS NOLIST NODETAILS;
  CONTENTS DATA=work.query OUT=work.details;
RUN;

PROC PRINT DATA=work.details;
RUN;

```

A new Program tab is created with the code that is used to create the view of the table. This code first creates a new data table in the **Work** library and then prints the data table. You can save this code for use later with the Save button on the toolbar.

9. Close the Query code. Exit the maximized view, and expand the **Columns** pane to get back to the default table view. You can clear the table filter by selecting **Clear Filter** on the Tools table.

End of Demonstration



Writing a Program in SAS Studio

Write a SAS program that enables you to see the **cars** data in the form of a report.

1. Enter the word **PROC** in the Program 1 workspace.

The screenshot shows the SAS Studio interface with the 'CODE' tab selected. In the code editor, the text '1proc' is typed. A context-sensitive help window is displayed, titled 'Global Statements'. It lists 'PROC' and 'PROCEDURE' under the 'Global Statements' category. The 'PROC' entry is highlighted. The help window provides the following information:

- Keyword:** [PROC](#)
- Context:** [GLOBAL STATEMENT] PROC statement
- Syntax:** PROC procedure-name <options>;
- Description:** Begins a PROC step. The PROC step consists of a procedure, usually with a SAS data set as input.
- Search:** [Product Documentation](#) [Samples and Examples](#)

As you begin to type, notice the context-sensitive Help, which is useful when you are learning SAS programming.

2. Enter the word **print** and notice how the context-sensitive Help changes.

The screenshot shows the SAS Studio interface with the 'CODE' tab selected. In the code editor, the text '1proc print' is typed. A context-sensitive help window is displayed, titled 'Procedures'. It lists 'PRINT' and 'PRINTTO' under the 'Procedures' category. The 'PRINT' entry is highlighted. The help window provides the following information:

- Keyword:** [PRINT](#)
- Context:** [PROCEDURE DEFINITION] PROC PRINT
- Syntax:** PROC PRINT <option(s)>;
BY <DESCENDING> variable-1 <...<DESCENDING> variable-n;
PAGEBY BY-variable;
SUMBY BY-variable;
- Options:** ID variable(s) <option>;

Scroll through the Context Help window. First notice the syntax for the PRINT procedure.

General form of the PRINT procedure:

```
PROC PRINT <option(s)>;
  BY <DESCENDING> variable-1 <...<DESCENDING>
    variable-n><NOTSORTED>;
  PAGEBY BY-variable;
  SUMBY BY-variable;

  ID variable(s) <option>;
  SUM variable(s) <option>;
  VAR variable(s) <option>;
```

Next, notice a description of the procedure.

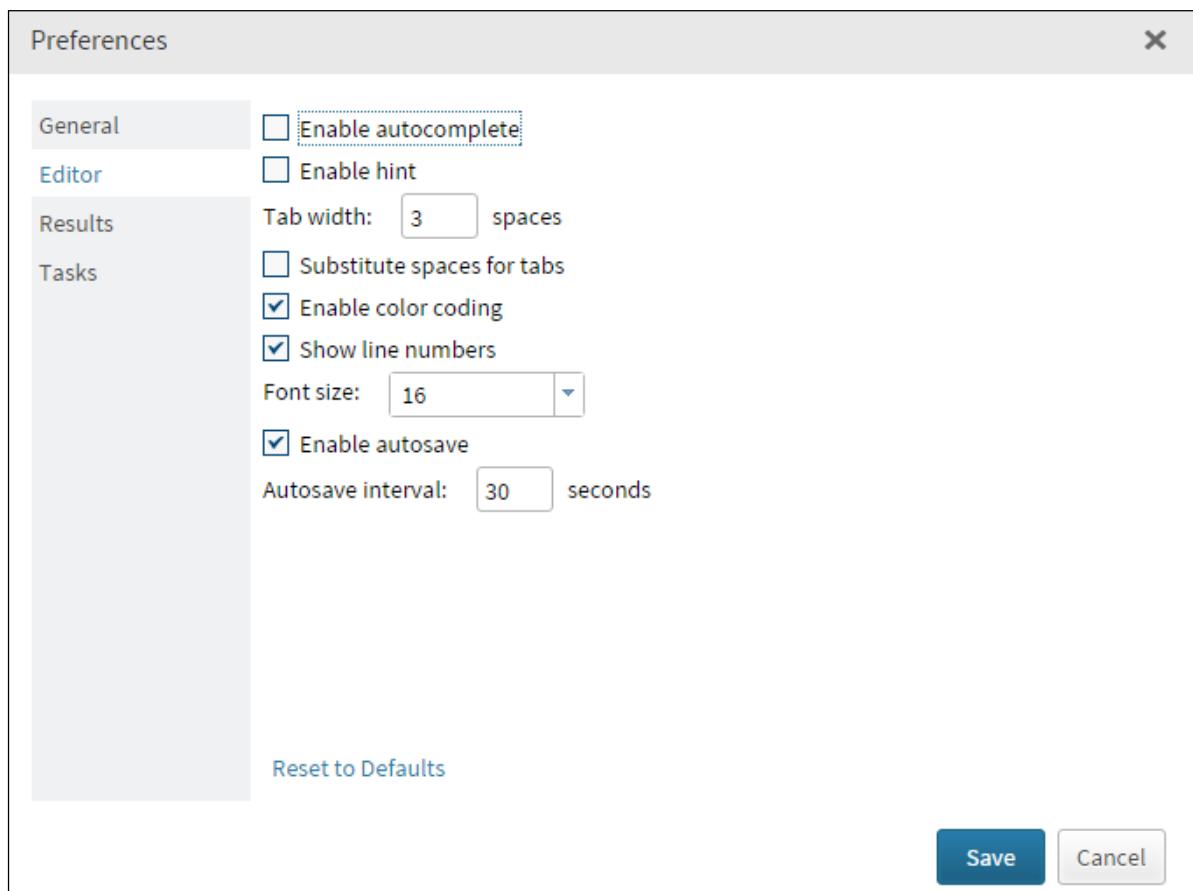
The PRINT procedure prints the observations in a SAS data set, using all or some of the variables. You can create a variety of reports ranging from a simple listing to a highly customized report that groups the data and calculates totals and subtotals for numeric variables.

Beginning in SAS 9.3, the PRINT procedure is now completely integrated with the Output Delivery System.

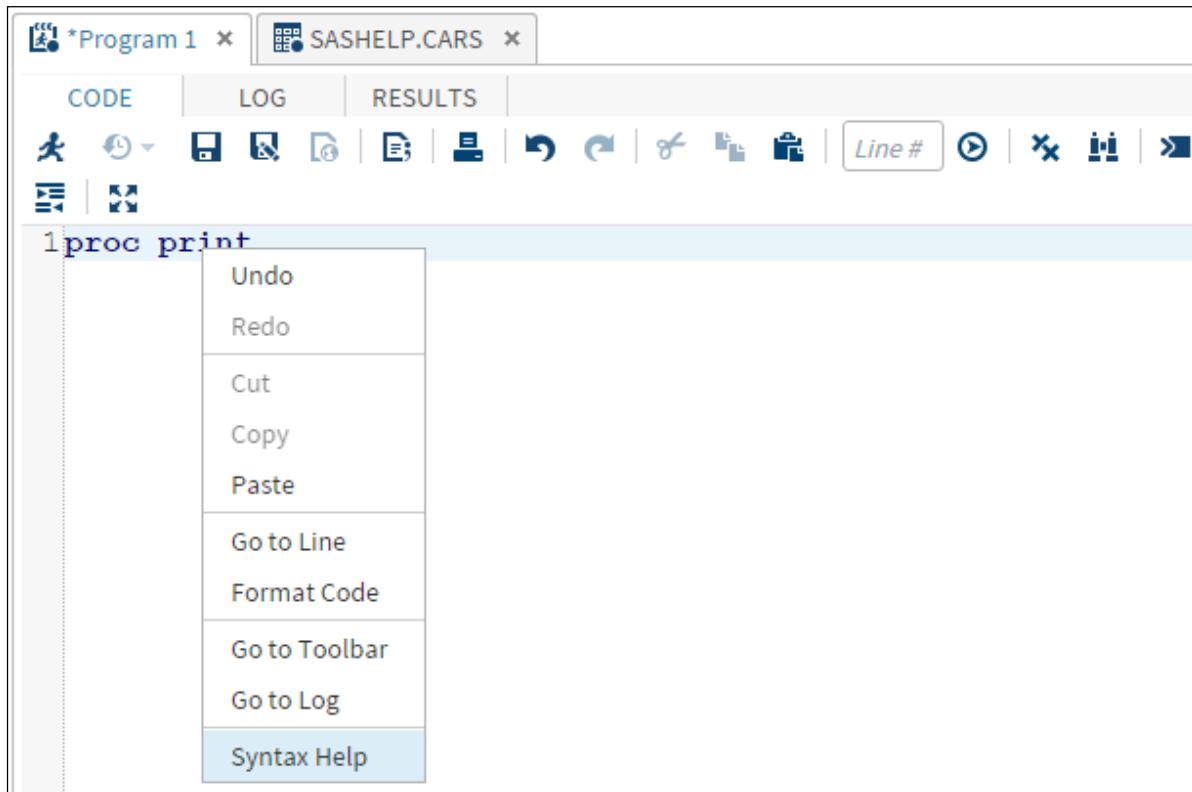
Finally, the context-sensitive Help provides links to SAS documentation and samples.

Search: **Product Documentation** ⇔ **Samples and SAS Notes** ⇔ **Papers**

3. To turn off the context Help, select **More Application Options** ⇒ **Preferences** ⇒ **Editor**. Clear the **Enable autocomplete** check box. Select **Save**.



4. To view the context Help without the Autocomplete option, right-click the keyword **print** and select **Syntax Help**.



5. Finish the program by entering the following code:

```
proc print data=sashelp.cars;
run;
```

This program tells SAS to print the data table **cars** in the **Sashelp** library. The DATA= option tells SAS which data set to use for the specified procedure. Notice that the library name is followed by a period and then the data set name.

6. Print the **cars** data table by clicking **Run** on the toolbar or pressing **F3**.

The screenshot shows the SAS Studio interface. On the left is a sidebar with 'Files and Folders', 'Tasks', 'Snippets', and 'Libraries' sections. The 'Libraries' section is expanded, showing 'My Libraries' and 'SASHHELP' with various datasets like '_CMPIDX_'. The main area has tabs for 'CODE', 'LOG', and 'RESULTS'. The 'RESULTS' tab is active, displaying a data table titled 'SASHHELP.CARS'. The table has columns: Obs, Make, Model, Type, Origin, DriveTrain, MSRP, Invoice, EngineSize, and Cylinders. The data consists of 14 rows of car information.

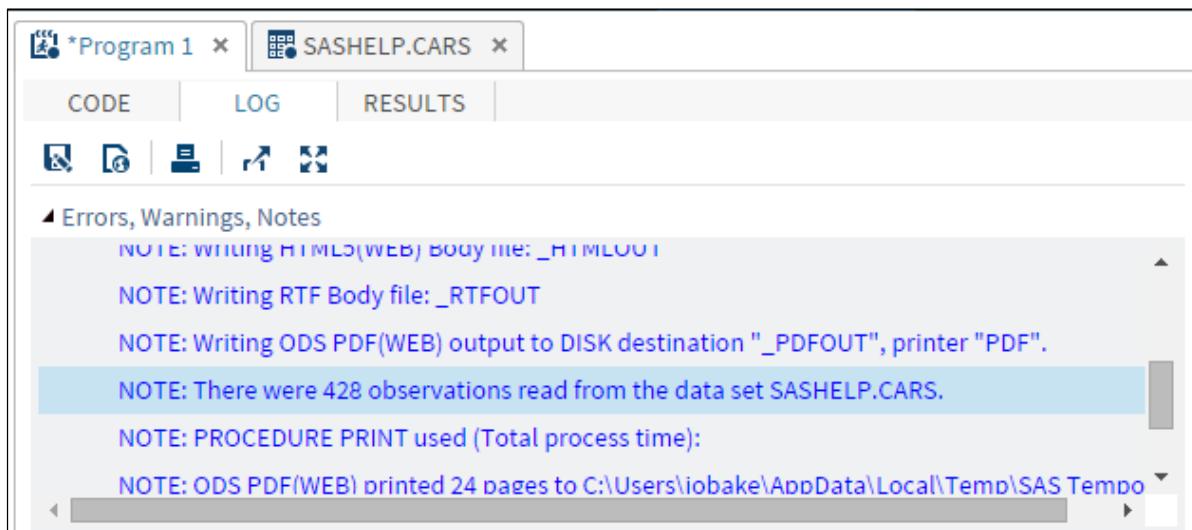
Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders
1	Acura	MDX	SUV	Asia	All	\$36,945	\$33,337	3.5	6
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	\$23,820	\$21,761	2.0	4
3	Acura	TSX 4dr	Sedan	Asia	Front	\$26,990	\$24,647	2.4	4
4	Acura	Tl 4dr	Sedan	Asia	Front	\$33,195	\$30,299	3.2	6
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	\$43,755	\$39,014	3.5	6
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	\$46,100	\$41,100	3.5	6
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	\$89,765	\$79,978	3.2	6
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	\$25,940	\$23,508	1.8	4
9	Audi	A41.8T convertible 2dr	Sedan	Europe	Front	\$35,940	\$32,506	1.8	4
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	\$31,840	\$28,846	3.0	6
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	\$33,430	\$30,366	3.0	6
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	\$34,480	\$31,388	3.0	6
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	\$36,640	\$33,129	3.0	6
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	\$39,640	\$35,992	3.0	6

The results are displayed on the RESULTS tab. Scroll to view different parts of the table. You can open the results in another window, by clicking the **Open in New Browser** tab. In addition, the toolbar on the Results page provides several ways to save the results. You can download and save the results in a Word, PDF, or HTML document by selecting the appropriate icon.

7. As a best practice, always click the **Log** tab to view the errors, warnings, and notes.

The screenshot shows the SAS Studio interface with the 'LOG' tab active. The title bar says '*Program 1 x SASHHELP.CARS x'. Below the tabs are icons for code, log, results, and other functions. Under the 'LOG' tab, there is a section titled 'Errors, Warnings, Notes' with three items: 'Errors', 'Warnings', and 'Notes (11)'.

8. Click the **Notes** arrow to view the 11 notes that were created.



The screenshot shows the SAS Log window with two tabs: 'CODE' and 'LOG'. The 'LOG' tab is selected. Below the tabs are several icons. The log content is as follows:

- NOTE: Writing HTML5(WEB) Body file: _HTMLOUT
- NOTE: Writing RTF Body file: _RTFOUT
- NOTE: Writing ODS PDF(WEB) output to DISK destination "_PDFOUT", printer "PDF".
- NOTE: There were 428 observations read from the data set SASHelp.CARS.** (This line is highlighted in blue.)
- NOTE: PROCEDURE PRINT used (Total process time):
- NOTE: ODS PDF(WEB) printed 24 pages to C:\Users\iobake\AppData\Local\Temp\SAS Tempo

Notice that the log reports that there were 428 observations read from the **sashelp.cars** data set.

 When the log reports errors, it is much easier to click the **Errors** arrow rather than searching for the error throughout the log.

9. Create a new program by selecting **New Options** at the top of the page and then selecting **New SAS Program** (or simply press **F4**).
10. Add the following code to the Program 2 workspace. Use the VAR statement to print only the desired column variables: **Make**, **Model**, **MPG_City**, and **MPG_Highway**.

```
proc print data=sashelp.cars;
  var
  run;
```

11. In the Libraries pane, select the arrow next to the **cars** data set to view the variables in the data set. Drag and drop the four variables into the program to complete the program.

```
proc print data=sashelp.cars;
  var Make Model MPG_City MPG_Highway;
  run;
```

 You can also enter the name of each variable.

12. Run the program and view the results.

The screenshot shows the SAS Studio interface with three tabs at the top: *Program 1*, SASHHELP.CARS, and *Program 2*. The *Program 2* tab is active. Below the tabs is a navigation bar with buttons for CODE, LOG, and RESULTS. The RESULTS tab is selected. Underneath is a toolbar with icons for copy, paste, cut, download, print, and refresh. The main area displays a data grid with the following columns: Obs, Make, Model, MPG_City, and MPG_Highway. The data consists of 22 rows, each representing a different car model from Acura and Audi, along with their respective MPG values.

Obs	Make	Model	MPG_City	MPG_Highway
1	Acura	MDX	17	23
2	Acura	RSX Type S 2dr	24	31
3	Acura	TSX 4dr	22	29
4	Acura	TL 4dr	20	28
5	Acura	3.5 RL 4dr	18	24
6	Acura	3.5 RL w/Navigation 4dr	18	24
7	Acura	NSX coupe 2dr manual S	17	24
8	Audi	A4 1.8T 4dr	22	31
9	Audi	A41.8T convertible 2dr	23	30
10	Audi	A4 3.0 4dr	20	28
11	Audi	A4 3.0 Quattro 4dr manual	17	26
12	Audi	A4 3.0 Quattro 4dr auto	18	25
13	Audi	A6 3.0 4dr	20	27
14	Audi	A6 3.0 Quattro 4dr	18	25
15	Audi	A4 3.0 convertible 2dr	20	27
16	Audi	A4 3.0 Quattro convertible 2dr	18	25
17	Audi	A6 2.7 Turbo Quattro 4dr	18	25
18	Audi	A6 4.2 Quattro 4dr	17	24
19	Audi	A8 L Quattro 4dr	17	24
20	Audi	S4 Quattro 4dr	14	20
21	Audi	RS 6 4dr	15	22
22	Audi	TT 1.8 convertible 2dr (coupe)	20	28
--	--	--	--	--

Notice that only the four variables specified in the VAR statement are printed on the Results page.

End of Demonstration



Using Tasks

In addition to features that make writing SAS code easier, SAS Studio also includes powerful point-and-click tasks that quickly generate reports and graphs.

1. Select **Tasks** in the Navigation pane and then expand **Tasks**.

The screenshot shows the SAS Studio interface. On the left, the **Navigation pane** is open, displaying the **Tasks** category. Under **Tasks**, several sub-categories are listed: **Data**, **Graph**, **Combinatorics and Probability**, **Statistics**, **High-Performance Statistics**, **Econometrics**, **Forecasting**, and **Data Mining**. Below these are **Utilities** (with **Import Data**, **Query**, and **SAS Program**), **Snippets**, **Libraries**, and **File Shortcuts**. On the right, the **RESULTS** tab is selected in the top navigation bar. Below it, a table titled "SASHelp.CARS" displays data for various car models, including columns for Obs, Make, Model, MPG_City, and MPG_Highway. The table contains 22 rows of data.

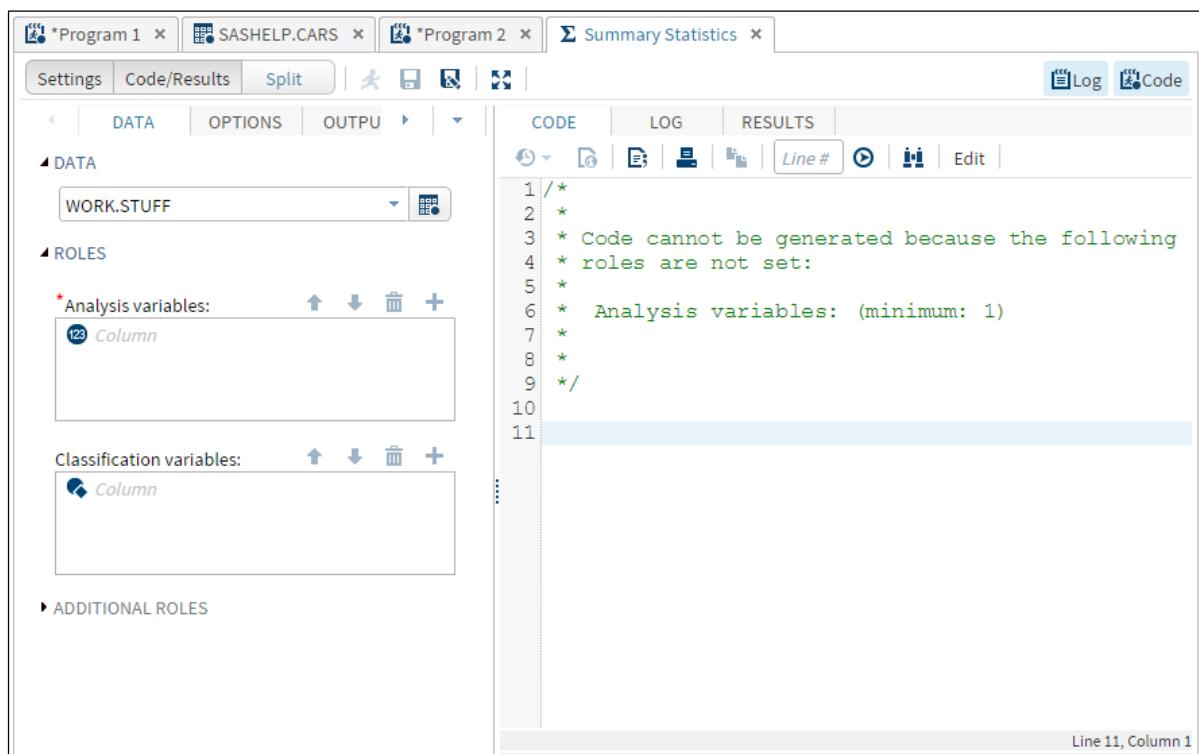
Obs	Make	Model	MPG_City	MPG_Highway
1	Acura	MDX	17	23
2	Acura	RSX Type S 2dr	24	31
3	Acura	TSX 4dr	22	29
4	Acura	TL 4dr	20	28
5	Acura	3.5 RL 4dr	18	24
6	Acura	3.5 RL w/Navigation 4dr	18	24
7	Acura	NSX coupe 2dr manual S	17	24
8	Audi	A4 1.8T 4dr	22	31
9	Audi	A41.8T convertible 2dr	23	30
10	Audi	A4 3.0 4dr	20	28
11	Audi	A4 3.0 Quattro 4dr manual	17	26
12	Audi	A4 3.0 Quattro 4dr auto	18	25
13	Audi	A6 3.0 4dr	20	27
14	Audi	A6 3.0 Quattro 4dr	18	25
15	Audi	A4 3.0 convertible 2dr	20	27
16	Audi	A4 3.0 Quattro convertible 2dr	18	25
17	Audi	A6 2.7 Turbo Quattro 4dr	18	25
18	Audi	A6 4.2 Quattro 4dr	17	24
19	Audi	A8 L Quattro 4dr	17	24
20	Audi	S4 Quattro 4dr	14	20
21	Audi	RS 6 4dr	15	22
22	Audi	TT 1.8 convertible 2dr (coupe)	20	28

Notice that the tasks are separated into the following categories based on the analysis:

- Data
- Graph
- Combinatorics and Probability
- Statistics
- High-Performance Statistics
- Econometrics
- Forecasting
- Data Mining

You can expand each node to view the possible tasks.

2. Expand the **Statistics** task and double-click the **Summary Statistics** task.



The screenshot shows the SAS Studio interface with the following details:

- Top Bar:** *Program 1*, SASHELP.CARS, *Program 2*, Summary Statistics.
- Left Panel (DATA):** WORK.STUFF selected under Analysis variables.
- Left Panel (ROLES):** Column selected under Analysis variables.
- Right Panel (CODE):** Displays generated code:

```
1 /*  
2 *  
3 * Code cannot be generated because the following  
4 * roles are not set:  
5 *  
6 * Analysis variables: (minimum: 1)  
7 *  
8 *  
9 */  
10  
11
```
- Status Bar:** Line 11, Column 1

Notice that a new tab opens with the title **Summary Statistics**.

3. In the Data section, click the **Select a Table** button and navigate to the **cars** data set in the **Sashelp** library.

The screenshot shows the SAS Studio interface with the 'DATA' tab selected. In the 'DATA' section, 'SASHHELP.CARS' is selected. Under 'Analysis variables:', 'Weight' is listed with a plus sign next to it, indicating it can be added. There are also sections for 'Classification variables:' and 'ADDITIONAL ROLES'.

4. Click the plus symbol next to Analysis variables and select **Weight** as the analysis variable.

The screenshot shows the SAS Studio interface with the 'DATA' tab selected. In the 'DATA' section, 'SASHHELP.CARS' is selected. Under 'Analysis variables:', 'Weight' is listed with a plus sign next to it, indicating it has been added. The right panel shows the generated SAS code:

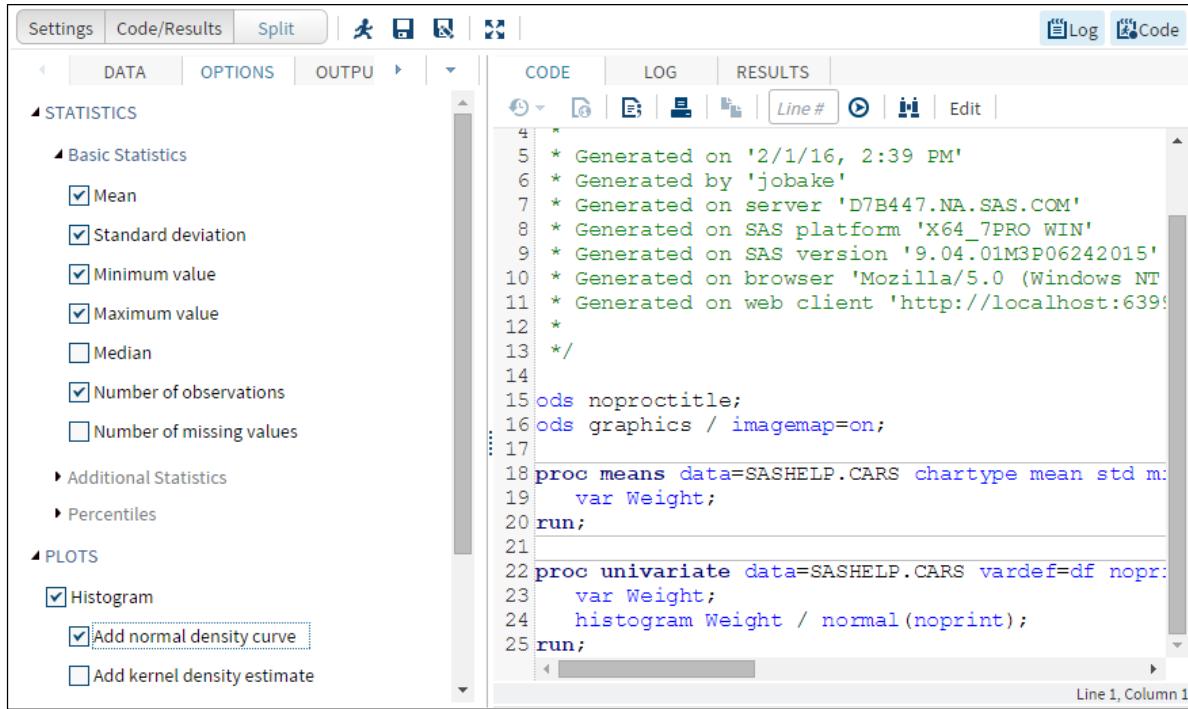
```

1 /* 
2 *
3 * Task code generated by SAS Studio 3.4
4 *
5 * Generated on '2/1/16, 2:37 PM'
6 * Generated by 'jobake'
7 * Generated on server 'D7B447.NA.SAS.COM'
8 * Generated on SAS platform 'X64_7PRO WIN'
9 * Generated on SAS version '9.04.01M3P06242015'
10 * Generated on browser 'Mozilla/5.0 (Windows NT 6
11 * Generated on web client 'http://localhost:63997
12 *
13 */
14
15 ods noproctitle;
16 ods graphics / imagemap=on;
17
18 proc means data=SASHHELP.CARS chartype mean std min
19   var Weight;
20 run;

```

Notice that SAS Studio automatically generates the code for the MEANS procedure.

5. Click the **OPTIONS** tab to specify which options you want to use. Ignore the Basic Statistics options. In the Plots section, select the **Histogram** and **Add normal density curve** check boxes to create statistical graphics.

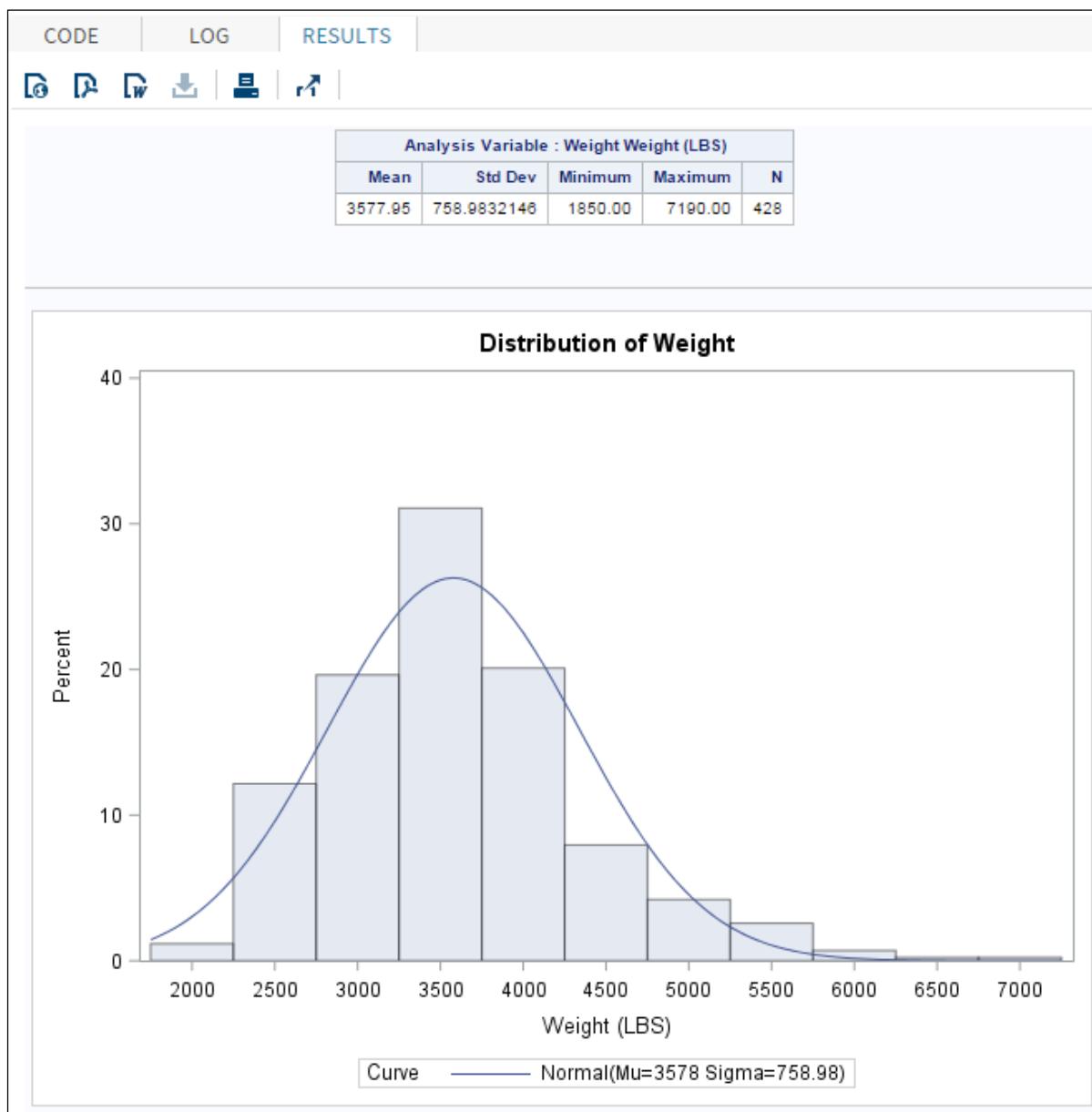


The screenshot shows the SAS Studio interface with the 'OPTIONS' tab selected in the top navigation bar. On the left, the 'STATISTICS' and 'PLOTS' sections are expanded, showing various options. Under 'PLOTS', the 'Histogram' and 'Add normal density curve' checkboxes are checked. The main area displays SAS code generated for these options:

```
4 *
5 * Generated on '2/1/16, 2:39 PM'
6 * Generated by 'jobake'
7 * Generated on server 'D7B447.NA.SAS.COM'
8 * Generated on SAS platform 'X64_7PRO WIN'
9 * Generated on SAS version '9.04.01M3P06242015'
10 * Generated on browser 'Mozilla/5.0 (Windows NT
11 * Generated on web client 'http://localhost:6399'
12 *
13 */
14
15 ods noproctitle;
16 ods graphics / imagemap=on;
17
18 proc means data=SASHELP.CARS chartype mean std m:
19   var Weight;
20 run;
21
22 proc univariate data=SASHELP.CARS vardef=df nopr:
23   var Weight;
24   histogram Weight / normal(noprint);
25 run;
```

Again, notice that SAS Studio automatically generates the code for the additional options.

6. Run the generated code and view the results.



The analysis is shown in a summary table and the plot is also printed on the Results page.

- ✎ You can save the program by clicking the **Save** button on the toolbar or by copying and pasting the code into an existing program.

End of Demonstration



Using Snippets

Code snippets enable you to quickly insert saved SAS code in your program and customize the code to meet your needs.

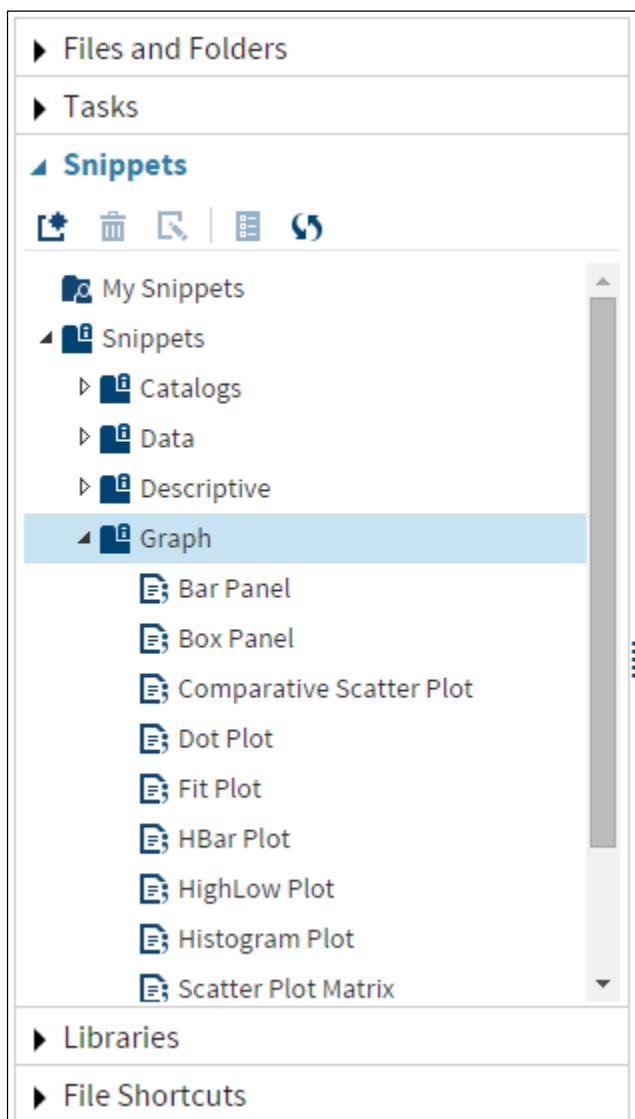
1. Open a new program tab by pressing **F4**.
2. In the Navigation pane, select **Snippets** and then expand the **Snippets** arrow.

A screenshot of the SAS Studio interface showing the Navigation pane. The pane is divided into several sections: 'Files and Folders' (with 'Tasks'), 'Snippets' (which is expanded to show 'My Snippets' and a folder named 'Snippets' containing categories like 'Catalogs', 'Data', 'Descriptive', 'Graph', 'IML', and 'Macro'). Other sections visible include 'Libraries' and 'File Shortcuts'. The 'Snippets' section is highlighted with a blue border.

SAS Studio has the following snippet categories preloaded for the user:

- Catalogs
- Data
- Descriptive
- Graph
- IML
- Macro

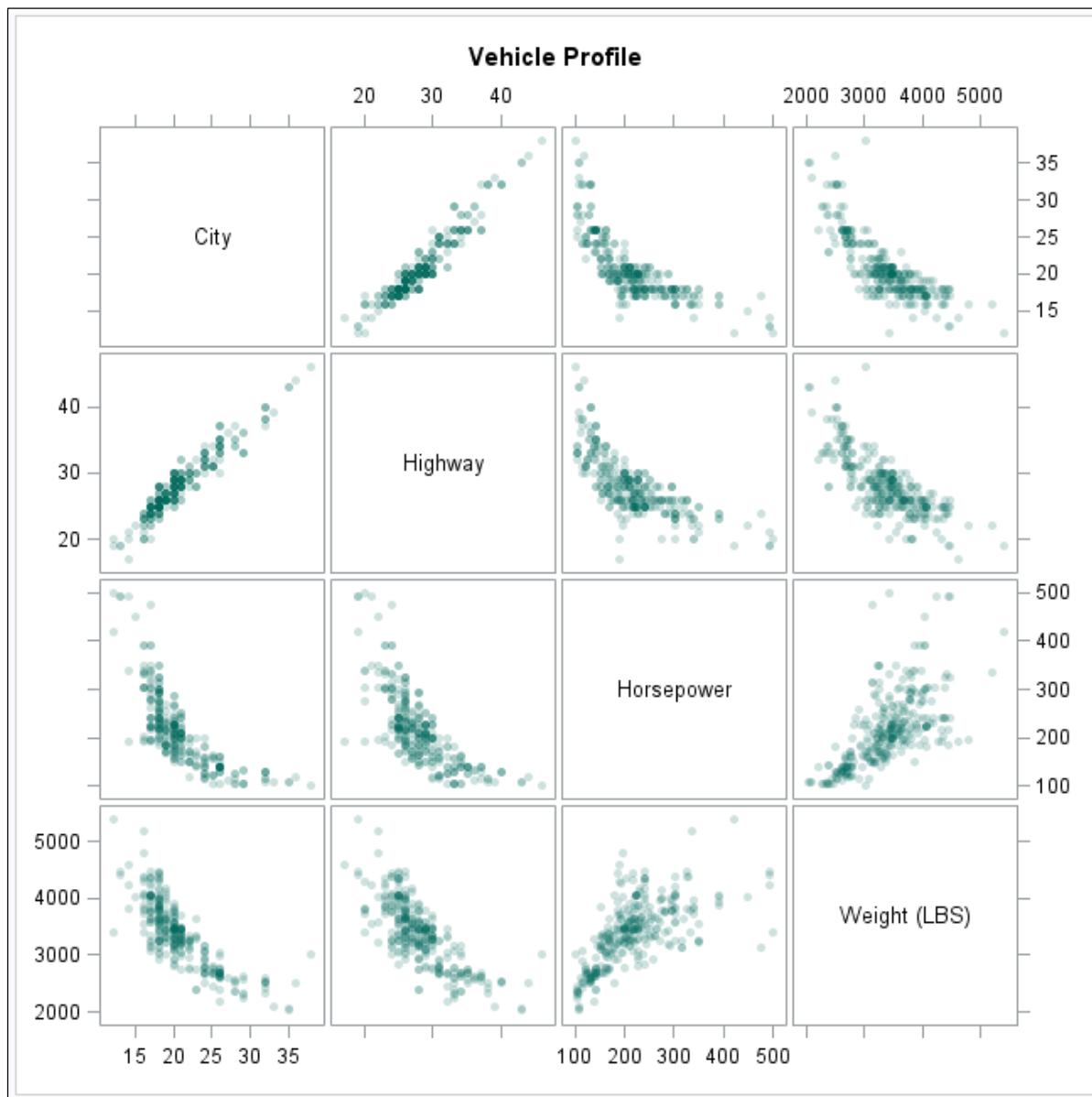
3. Expand **Graph**.



4. Drag and drop the **Scatter Plot Matrix** snippet into the program workspace. The following code is generated:

```
/*--Scatter Plot Matrix--*/  
  
title 'Vehicle Profile';  
proc sgscatter data=sashelp.cars(where=(type in ('Sedan' 'Sports')));  
  label mpg_city='City';  
  label mpg_highway='Highway';  
  matrix mpg_city mpg_highway horsepower weight /  
    transparency=0.8 markerattrs=graphdata3(symbol=circlefilled);  
run;
```

5. Click **Run** and view the results.

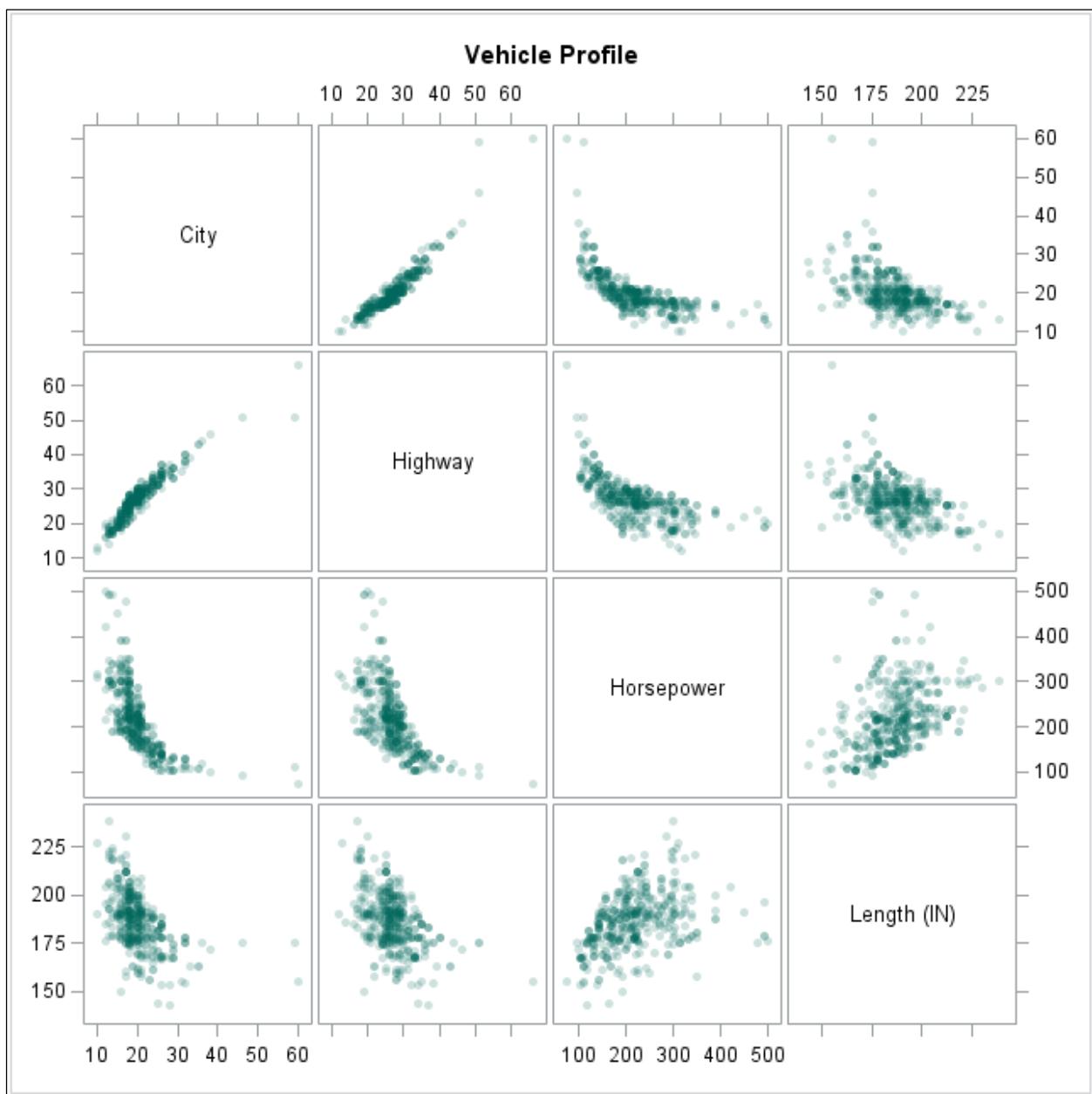


Generally, snippets are used as a starter program. Thus, the generated code can be altered to fit your needs.

6. Delete the WHERE option and change the **Weight** variable to the **Length** variable.

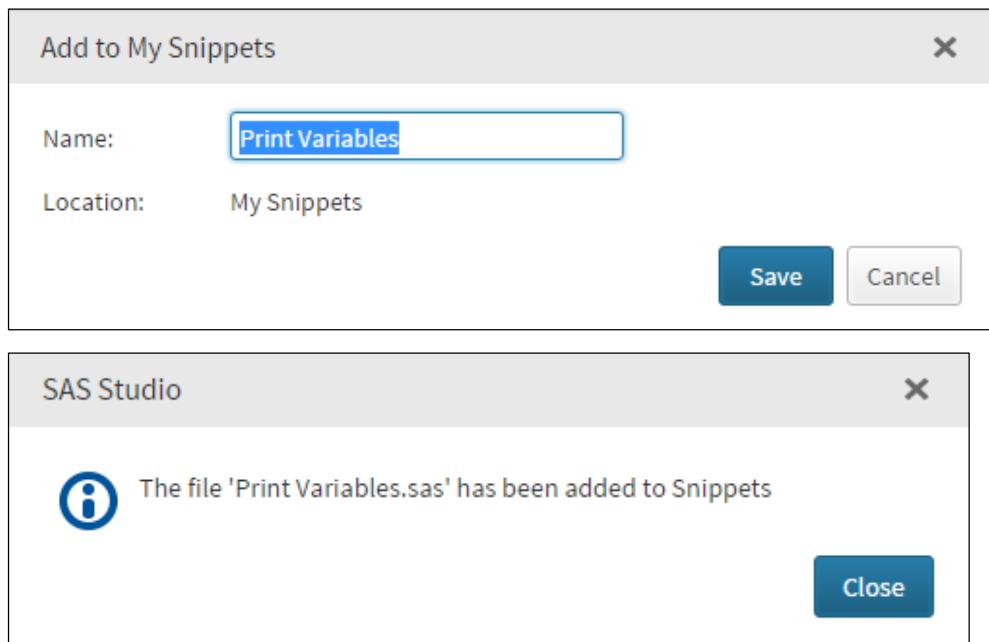
```
/*--Scatter Plot Matrix--*/
title 'Vehicle Profile';
proc sgscatter data=sashelp.cars;
label mpg_city='City';
label mpg_highway='Highway';
matrix mpg_city mpg_highway horsepower length /
transparency=0.8 markerattrs=graphdata3(symbol=circlefilled);
run;
```

7. Click **Run** and view the results.

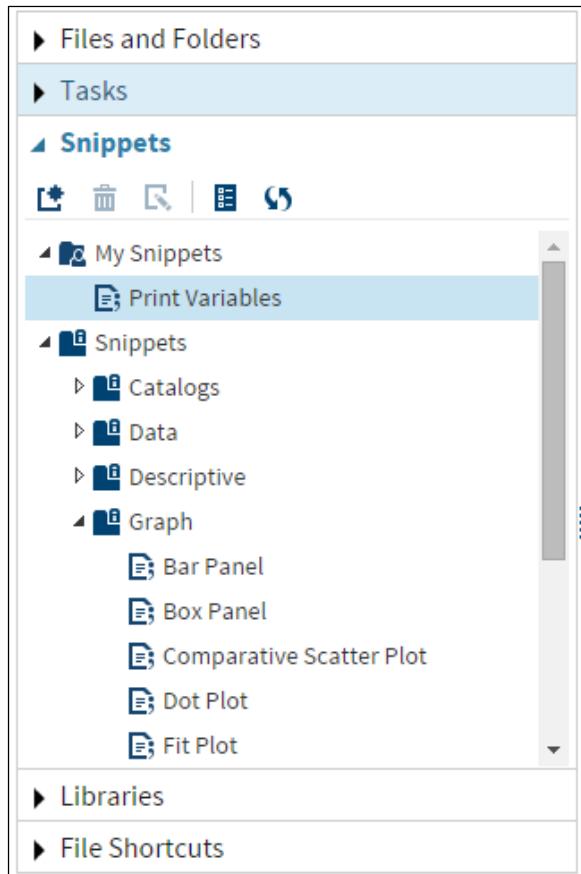


8. Create your own snippet by clicking the **New Snippet** button in the Snippets pane.
9. Copy and paste the SAS code from the Program 2 tab onto the Snippet 1 tab.
10. Click **Save** on the Snippet 1 tab.

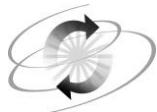
11. In the Add to My Snippets window, type **Print Variables** and click **Save**.



Notice that the My Snippets section now has the Print Variables snippet, which the user can drag and drop onto any SAS Studio Program tab at his convenience.



End of Demonstration



Exercises

1. Exploring the SAS Studio Interface

- a. Explore the **Sashelp** library.
- b. Query the data.
- c. Practice using the SAS Studio tasks.

You often use the **cars** data set throughout the course. Familiarize yourself with this data set.

The data set **cars** contains information about a sample of 1993 model cars from the *1993 Cars Annual Auto Issue* published by *Consumer Reports* and from *Pace New Car and Truck 1993 Buying Guide*. The data set consists of the following variables:

Make	Name of the manufacturer
Model	Name of the model
Type	Vehicle type (Hybrid, SUV, Sedan, Sports, Truck, or Wagon)
Origin	Vehicle origin (Asia, Europe, or USA)
DriveTrain	Drivetrain type (All, Front, or Rear)
Invoice	Invoice price
MSRP	Manufacturer's suggested retail price
EngineSize	Engine displacement size in liters
Cylinders	Number of Cylinders
Horsepower	Maximum horsepower
MPG_City	Average city miles per gallon (EPA rating)
MPG_Highway	Average highway miles per gallon (EPA rating)
Weight	Weight of vehicle in pounds
Wheelbase	Wheelbase in inches
Length	Length of the vehicle in inches

End of Exercises



Bayesian Logistic Regression

SP4R01d01.sas

Babies with low birth weights (defined to be less than 2500 grams) are a concern because of their potential medical problems. Health researchers want to identify possible contributing factors to low birth weight and recommend strategies to reduce the number of low birth weight babies. The data is named **birth** and includes the following variables:

ID	Identification code
LOW	Low birth weight (0 = birth weight \geq 2500g and 1 = birth weight $<$ 2500g)
AGE	Age of mother in years
LWT	Weight in pounds at the last menstrual period
ETH	Ethnicity
SMOKE	Smoking status during pregnancy (1 = Yes, 0 = No)
PTL	History of premature labor (0 = None, 1 = One, and so on)
HT	History of hypertension (1 = Yes, 0 = No)
UI	Presence of uterine irritability (1 = Yes, 0 = No)
FTV	Number of physician visits during the first trimester (0 = None, 1 = One, and so on)
BWT	Birth weight in grams

1. Read in the **birth** data set. Use a DATA step and create formats and labels to create a more informative analysis.

Partial DATA Step Code

```

proc format;
  value yesnofmt
    0="No"
    1="Yes";
  value ftvfmt
    0="0"
    1="1"
    2-high="2+";
  value ptlfmt
    0="0"
    1-high="1+";
run;

/*
LIST OF VARIABLES:

Columns      Variable                               Abbreviation
-----        -----
2-4          Identification Code                   ID
10           Low Birth Weight (0 = Birth Weight >= 2500g,
                           1 = Birth Weight < 2500g)       LOW

```

17-18	Age of the Mother in Years	AGE
23-25	Weight in Pounds at the Last Menstrual Period	LWT
32	Ethnicity	ETH
40	Smoking Status During Pregnancy (1 = Yes, 0 = No)	SMOKE
48	History of Premature Labor (0 = None 1 = One, etc.)	PTL
55	History of Hypertension (1 = Yes, 0 = No)	HT
61	Presence of Uterine Irritability (1 = Yes, 0 = No)	UI
67	Number of Physician Visits During the First Trimester (0 = None, 1 = One, 2 = Two, etc.)	FTV
73-76	Birth Weight in Grams	BWT

```

*/
```

```

data work.birth;
  input ID LOW AGE LWT ETH SMOKE PTL HT UI FTV BWT;
  if FTV>2 then FTV=2;
  if PTL>1 then PTL=1;
  label
    ID="ID Code"
    LOW="Birth Weight < 2500 Grams"
    AGE="Mom's Age"
    LWT="Mom's Weight Last Menstrual Period"
    ETH="Ethnicity"
    SMOKE="Smoking Status"
    PTL="Hx of Premature Labor"
    HT="Hx of Hypertension"
    UI="Hx of Uterine Irritability"
    FTV="MD Visits 1st Trimester"
    BWT="Birth Weight, Grams"
  ;
  format LOW SMOKE HT UI yesnofmt. PTL ptlfmt. ftv ftvfmt. ;
  datalines;
85   0      19     182    2      0      0      0      1      0      2523
86   0      33     155    3      0      0      0      0      3      2551
...
;
run;
```

2. Generate summary statistics for the **BWT** variable along with a histogram and QQPlot.

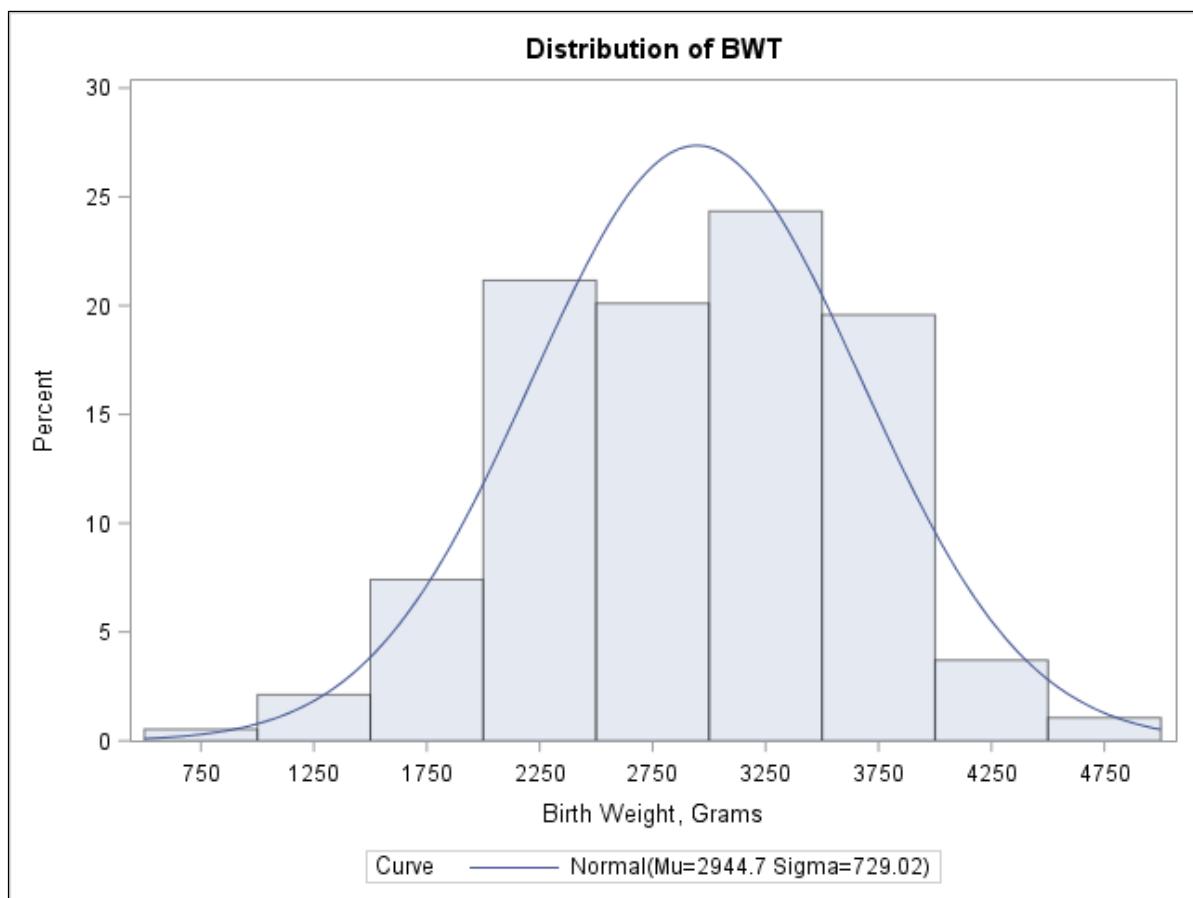
```
ods select basicmeasures histogram qqplot;
proc univariate data=work.birth;
  var bwt;
  histogram bwt / normal(mu=est sigma=est);
  qqplot bwt / normal(mu=est sigma=est);
run;
```

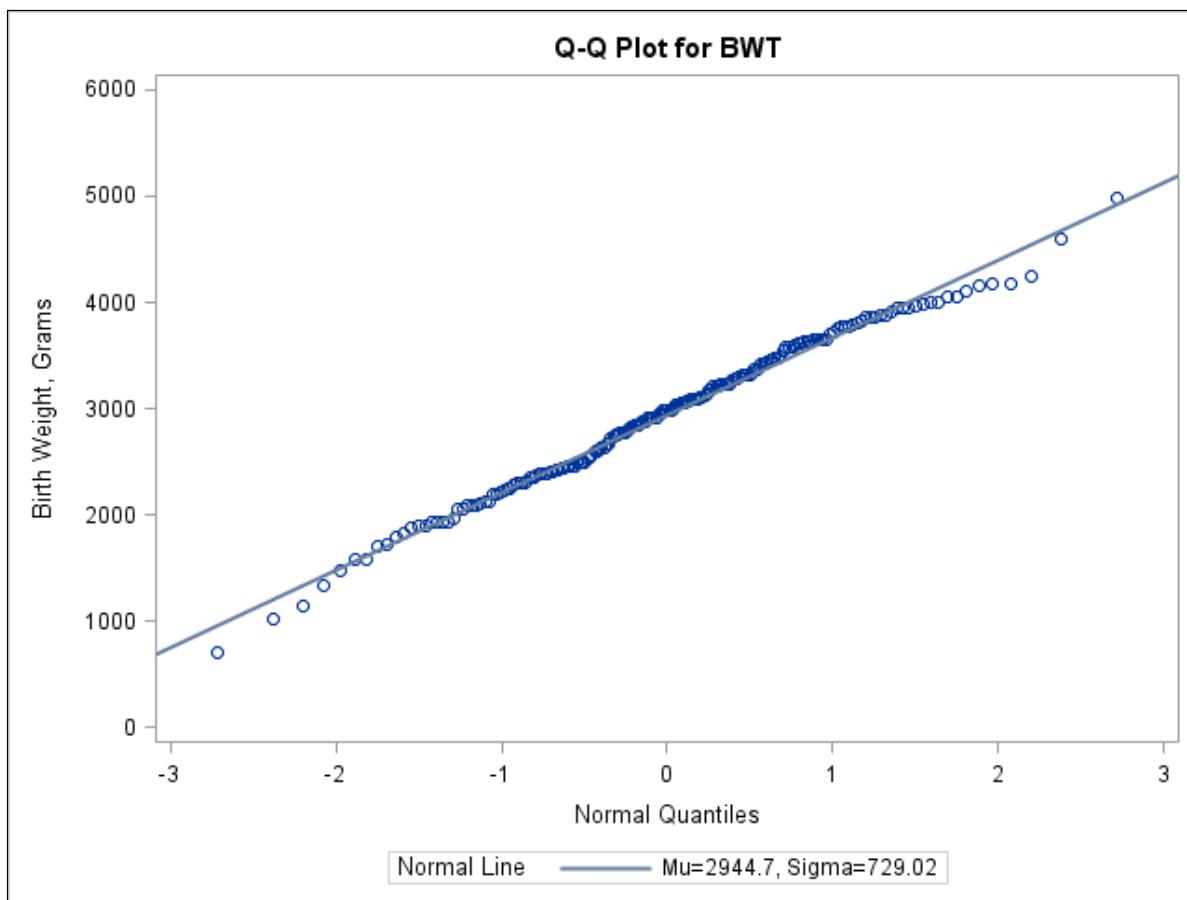
The UNIVARIATE Procedure
Variable: BWT (Birth Weight, Grams)

Basic Statistical Measures

	Location	Variability	
Mean	2944.656	Std Deviation	729.02242
Median	2977.000	Variance	531474
Mode	2495.000	Range	4281
		Interquartile Range	1061

Note: The mode displayed is the smallest of 4 modes with a count of 4.





3. Create univariate contingency tables for the variables **Low**, **Smoke**, **HT**, and **PTL**.

```
proc freq data=work.birth;
  table low smoke ht ptl;
run;
```

The FREQ Procedure

Birth Weight < 2500 Grams

LOW	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	130	68.78	130	68.78
Yes	59	31.22	189	100.00

Smoking Status

SMOKE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	115	60.85	115	60.85
Yes	74	39.15	189	100.00

Hx of Hypertension				
HT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	177	93.65	177	93.65
Yes	12	6.35	189	100.00
Hx of Premature Labor				
PTL	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	159	84.13	159	84.13
1+	30	15.87	189	100.00

4. Use the SAS MCMC procedure (Markov Chain Monte Carlo) to create a Bayesian logistic regression model with LOW as the dependent variable and SMOKE, HT, LWT, and PTL as the independent variables.

```
ods select nobs parameters postsummaries postintervals autocorr
tadpanel;
proc mcmc data=work.birth outpost=birthout diag=all dic propcov=quanew
nbi=5000 ntu=5000 nmc=200000 thin=10 mchistory=brief
plots(smooth)=all seed=27513 stats=all;
parms (beta0 beta1 beta2 beta3 beta4) 0;
prior beta: ~ normal(0, var=100);
p=logistic(beta0+beta1*smoke+beta2*ht+ beta3*lwt+beta4*ptl);
model low ~ binary(p);
title "Bayesian Analysis of Low Birth Weight Data";
run;
title;
```

The PARMS statement specifies the five parameters with initial values of 0. The PRIOR statement specifies a normal prior distribution with a mean of 0 and a variance of 100 for each parameter. The *p* assignment statement computes the probability of low birth weight using the parameter estimates, data values, and the logit link transformation (with the SAS function LOGISTIC). The MODEL statement specifies that the response variable **low** has a binary distribution with a parameter *p*.

Selected PROC MCMC statement options:

- OUTPOST= specifies an output data set that contains the posterior samples of all model parameters.
- PROPCOV= specifies the method used in constructing the initial covariance matrix for the Metropolis-Hastings algorithm. The quasi-Newton optimization (QUANEW) and the Nelder-Mead simplex optimization (NMSIMP) methods find numerically approximated covariance matrices at the optimum of the posterior density function with respect to all continuous parameters. The optimization does not apply to discrete parameters. The tuning phase starts at the optimized values. In some problems, this can greatly increase convergence performance.
- NTU= specifies the number of iterations to use in each proposal tuning phase. By default, NTU=500.
- NMC= specifies the number of iterations in the main simulation loop. This is the MCMC sample size if there is no thinning. By default, NMC=1000.

- THIN=n** controls the thinning rate of the simulation. PROC MCMC keeps every nth simulation sample and discards the rest. All of the posterior statistics and diagnostics are calculated using the thinned samples. By default, THIN=1.
- NBI=n** specifies the number of burn-in iterations to perform before beginning to save parameter estimate chains. By default, NBI=1000.
- MCHISTORY=** controls the display of the Markov chain sampling history. The keyword BRIEF produces a summary output for the tuning, burn-in, and sampling history tables.
- STATS=** specifies options for posterior statistics. You can request all of the posterior statistics by specifying STATS=ALL. You can suppress all the calculations by specifying STATS=NONE.

Partial PROC MCMC Output

The MCMC Procedure				
Number of Observations Read				189
Number of Observations Used				189
Parameters				
Block	Parameter	Sampling Method	Initial Value	Prior Distribution
1	beta0	N-Metropolis	0	normal(0, var=100)
	beta1		0	normal(0, var=100)
	beta2		0	normal(0, var=100)
	beta3		0	normal(0, var=100)
	beta4		0	normal(0, var=100)

The first table that PROC MCMC produces is the Number of Observations table. This table lists the number of observations read from the input data set and the number of nonmissing observations used in the analysis. The Parameters table lists the names of the parameters, the blocking information, the sampling method used, the starting values, and the prior distributions. You should check this table to ensure that you have specified the parameters correctly, especially for complicated models.

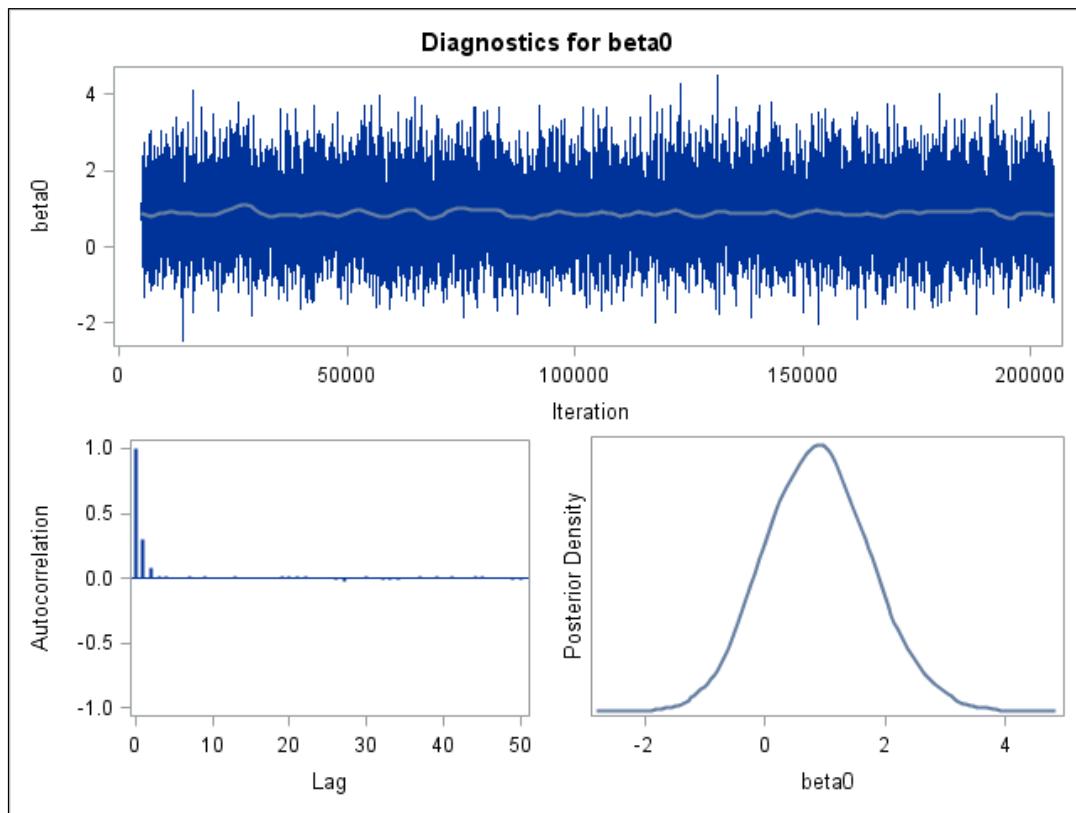
Posterior Summaries						
Parameter	N	Mean	Standard Deviation	Percentiles		
				25	50	75
beta0	20000	0.8812	0.8756	0.2758	0.8693	1.4598
beta1	20000	0.5124	0.3461	0.2785	0.5111	0.7460
beta2	20000	1.9292	0.7443	1.4288	1.9108	2.4112
beta3	20000	-0.0179	0.00685	-0.0224	-0.0178	-0.0132
beta4	20000	1.3191	0.4447	1.0220	1.3155	1.6145

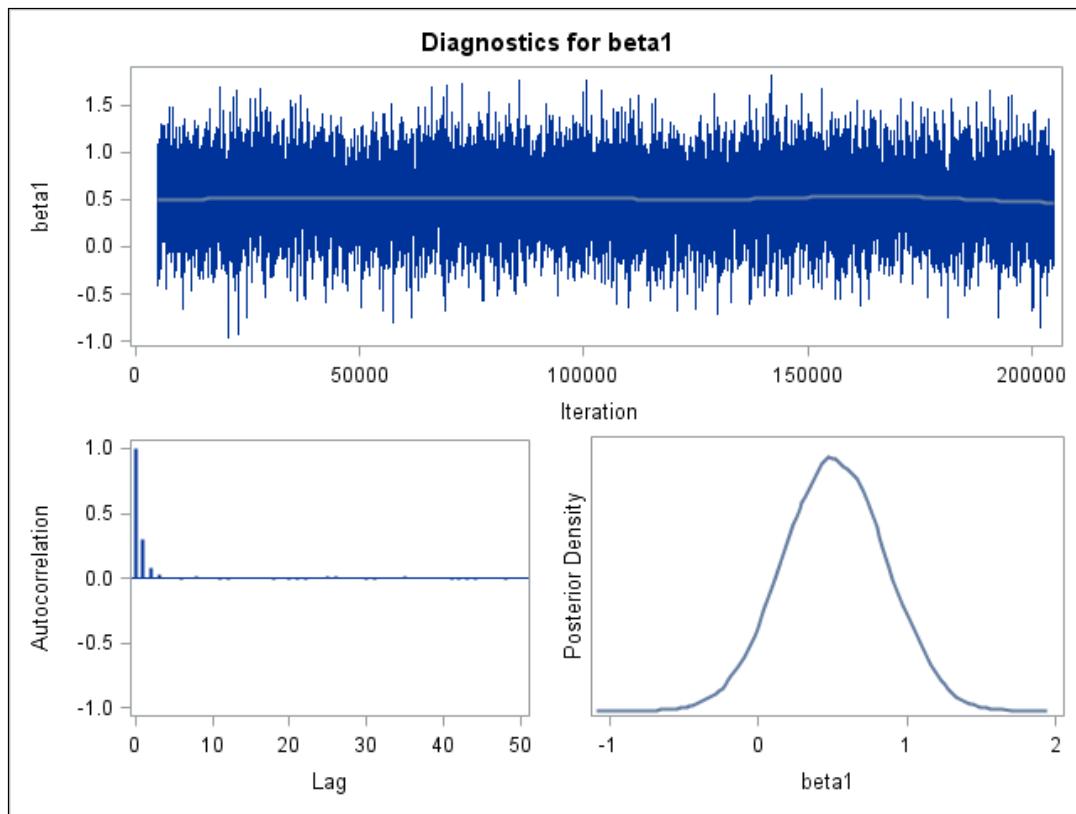
Posterior Intervals					
Parameter	Alpha	Equal-Tail Interval		HPD Interval	
beta0	0.050	-0.7734	2.6647	-0.8075	2.6183
beta1	0.050	-0.1665	1.1824	-0.1533	1.1941
beta2	0.050	0.5218	3.4477	0.4763	3.3879
beta3	0.050	-0.0321	-0.00522	-0.0317	-0.00493
beta4	0.050	0.4448	2.1960	0.4384	2.1874

For each posterior distribution, PROC MCMC also reports summary statistics (posterior means and standard deviations) and interval statistics (95% highest posterior density credible intervals).

Posterior Autocorrelations				
Parameter	Lag 1	Lag 5	Lag 10	Lag 50
beta0	0.2951	-0.0036	-0.0017	-0.0118
beta1	0.2993	-0.0027	0.0011	0.0055
beta2	0.3186	-0.0037	-0.0103	-0.0190
beta3	0.2975	-0.0037	-0.0060	-0.0134
beta4	0.3199	-0.0070	0.0058	0.0007

Partial Graphics Output





End of Demonstration



Poker Simulation

This program shows how you can generate poker hands for Texas Hold ‘Em. For simplicity, this program assumes nine players.

1. The beginning of the program specifies the number of hands to be generated. A standard deck of cards has 13 ranks (ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, and king) and four suits (clubs, diamonds, hearts, and spades). Create the deck as the combination of every rank with every suit and print the deck.

```
proc iml;
nHands=10; *Specify number of hands;

*Create deck;
rank={A,2,3,4,5,6,7,8,9,10,J,Q,K};
suit={C D H S}; *Clubs diamonds hearts spades;
deck=concat(right(repeat(rank,1,4)),repeat(suit,13,1));
print deck;
```

deck			
AC	AD	AH	AS
2C	2D	2H	2S
3C	3D	3H	3S
4C	4D	4H	4S
5C	5D	5H	5S
6C	6D	6H	6S
7C	7D	7H	7S
8C	8D	8H	8S
9C	9D	9H	9S
10C	10D	10H	10S
JC	JD	JH	JS
QC	QD	QH	QS
KC	KD	KH	KS

2. In Texas Hold ‘Em, the dealer provides two face-down cards to each player and deals five face-up community cards. For a nine-player game, the dealer deals a total of 23 cards for each hand. Use the SAMPLE function to generate a sample of 23 cards for each of **nhands** hands without replacement.

```
*Sample cards from deck;
cards=sample(deck,(23//nhands),"WOR");
```

3. To simplify the results of the simulation, combine the first and second card dealt to each person in each hand into a single cell. This reduces the number of columns in the hands matrix from 23 to 14 (for a simulation of nine players).

```
*Combine 1st and 2nd card for each person into single cell;
card1=cards[,1:9];
card2=cards[,10:18];
community=cards[,19:23];
hands=concat(card1,",",card2) || community;
```

4. Create a vector of column names.

```
*Create column names;
do i=1 to 9;
    name=name || ("p"+strip(char(i)));
end;

name=name || {c1 c2 c3 c4 c5};
```

5. Finally, print the first 10 hands from the **names** matrix.

```
print (hands[1:10,]) [colname=name];
quit;
```

	p1	p2	p3	p4	p5	p6	p7	p8	p9	C1	C2	C3	C4	C5
ROW1	7D, 7S	AS, 6S	JS, AC	4C, 2C	6H, QC	6C, 3C	5C, KD	JC,10H	4D, 2S	3H	8C	10C	8H	AH
ROW2	QH, 4H	4S, KC	6S,10H	3H, 8S	10D, 2D	2S, KS	10C, AD	QD, JH	8D, 7C	4D	3D	KD	3C	8H
ROW3	4D, JC	AD, JD	9S, 7S	AS, 4H	2S, 9D	2C, 5H	3S, 5C	6C, 5S	9C, 7C	10C	3H	6D	KD	KH
ROW4	4H, 3S	7D, 4D	KS, 6C	AC, KC	JS, 7H	9H, QH	7C, QD	6D, 4S	8H, 2C	10C	2H	5D	KH	AH
ROW5	2C, AD	3C, 9C	5C, 9H	10C, 6S	7S, 2D	10D, KH	3H, 4H	5H, QC	JH, 9S	JC	KC	KD	4S	8D
ROW6	9H, 3H	JH, 5S	2S, QH	7D, 7S	JS,10C	KC, 6H	AD, 2C	KH, 3D	7H, 9S	5D	AH	2H	6D	3C
ROW7	7S, 3S	9C, 6H	5D, 8C	5C, 3D	4S, 3H	6C, QC	8D, JS	2D, QH	KD, 9H	10D	AD	5S	3C	10S
ROW8	7S,10C	6D, KC	10H, JS	QS, KH	3D, 9C	QH, AS	KS, JH	3S, JC	8S, AD	2D	3H	6H	4S	2C
ROW9	6H,10D	QH, AC	3S, 8D	6C, 2H	7C, AS	JD, 9D	2S,10S	8C, 7H	JH, 2D	QD	4C	2C	4H	KH
ROW10	KS, 5C	9C, 4D	5S, JC	3C, 8H	5H, QD	AD, 3H	9D, 2D	KD,10S	QH, 4C	3S	JD	4H	2H	AH

6. Find the probability of the best starting hand in Texas Hold ‘Em, the illusive pocket aces. To do so, create a new deck that ignores the suit and holds only the rank of the 52 cards. Next, set the number of hands to be simulated to 10,000.

```
deck = repeat(rank, 4);
hands=10000;
```

7. Create a loop that simulates the 10,000 hands and initialize the variable **Count** to 0. If any hand inside the loop is dealt pocket aces, increment the **Count** variable by 1.

```
*Sample many hands and count the number of pocket aces;
count = 0;
do i=1 to hands;
    sam = sample(deck,2,"WOR");
```

```
aces= (sam[1]='A' & sam[2]='A') ;
if aces=1 then do;
print sam;
count=count+1;
end;
end;
```

8. Find the probability of pocket aces by dividing the **Count** variable by the number of hands and print the result.

```
p=count/hands;
print count hands p;
quit;
```

count	hands	p
48	10000	0.0048

Notice that of the 10,000 simulations, only 48 hands were dealt two aces. This gives an empirical estimate of 0.0048 for pocket aces.

End of Demonstration



Determining Power Using Simulation

This program shows how you might obtain an empirical estimate of power for a coefficient in a multiple regression analysis using simulation.

Consider a researcher who wants to determine the effect of independent variable X on dependent variable Y , controlling for covariates C1 and C2. A literature search indicates that the predictor variables ($X, C1,$

and $C2$) have the following correlation matrix: $\begin{bmatrix} 1 & .2 & .2 \\ .2 & 1 & .2 \\ .2 & .2 & 1 \end{bmatrix}$. The literature indicates that the variances

of the predictors are $\begin{bmatrix} 5 \\ 3 \\ 7 \end{bmatrix}$. The means of the predictors are $\begin{bmatrix} 20 \\ 5 \\ 10 \end{bmatrix}$. The regression coefficients are

expected to be $\begin{bmatrix} 1 \\ 2 \\ .5 \end{bmatrix}$ and the residual variance is expected to be 3.7. The researcher plans to collect

a sample of size $n=30$, and wants to know her power to detect a significant effect for the partial regression coefficient predicting Y from X at alpha equals .05.

1. Specify the number of replications to be used for the simulation and the random number seed.

```
proc iml;
  nRep=1000;
  call randseed(112358);
```

2. Enter the study parameters (sample size, mean vector, correlation matrix, variance vector, beta coefficients, and residual variance).

```
n=30;

mean={20,
      5,
      10};

corr={1 .2 .2,
      .2 1 .2,
      .2 .2 1};

var={5,
     3,
     7};

beta={ 1,
       2,
       .5};

resvar=14;
```

3. Before simulating the predictor variables, you need to calculate the covariance matrix for the predictor variables from the variance vector and correlation matrix. Create a matrix called **sddiag**, which is a diagonal matrix containing the standard deviations of the variables on the diagonal. Next, calculate the covariance matrix by pre-multiplying and post-multiplying the correlation matrix by **sddiag**.

```
sddiag=diag(sqrt(var));
cov=sddiag * corr * sddiag;
```

4. The distribution of the predictor variables is determined by the mean vector and covariance matrix that you entered earlier. Notice that the number of observations that you want to draw is specified as (**n * nrep**). Rather than using a loop to draw 30 observations at a time for each of 1000 iterations, you draw all 30000 observations in a single call of the RANDNORMAL function. This helps improve the efficiency of the program.

```
x=randNormal(n*nRep,mean,cov);
```

5. The vector of predicted scores for the dependent variable, **ypred**, is calculated as the matrix of predictor variables post-multiplied by the vector of beta coefficients. The error values are drawn from a univariate normal distribution centered at zero with a spread equal to the square root of **resvar** (the residual standard deviation). The dependent variable is calculated as the sum of the predicted scores and the errors.

```
yPred=x * beta;
error=randfun(n*nrep,"Normal",0,sqrt(resvar));
y=yPred + error;
```

6. Matrix **x** and vector **y** contain the simulated values for the independent and dependent variables, respectively, across all iterations. They do not indicate which observations correspond to which iterations. The column variable indicating iteration needs to contain the number 1 listed 30 times followed by the number 2 listed 30 times, and so on. This vector could easily be created in a loop, as shown in the commented code above. Alternatively, you can improve the efficiency by removing the loop and using a combination of REPEAT and COLVEC statements. The statement
temp=repeat((1:nrep)` ,1,n); creates an **n x nrep** (in this case, 30 x 1000) matrix named **temp**. Each column contains the numbers 1 through **nrep**. The statement
iteration=colvec(temp); converts each row to a column vector and then stacks the column vectors. The final result is **iteration**, a 30000 x 1 column vector.

```
temp=repeat((1:nrep)` ,1,n);
iteration=colvec(temp);
```

7. Combine the iteration numbers, the predictor variable values, and the dependent variable values into a single matrix for output. Then output the matrix to a data set called **temp** and store all of the matrices for later use.

```
sampleData=iteration || x || y;

create temp from sampleData [colname={iteration x c1 c2 y}];
append from sampleData;
close temp;

store;
quit;
```

8. The GLM procedure can be used to perform regression analyses. The DATA command specifies our newly created **temp** data set as the data set for analysis. The NOPRINT option suppresses the printing of results. OUTSTAT= outputs the results from GLM to a data set called **regResults**. The BY statement requests that a separate analysis be performed for each value of **iteration**. The MODEL statement specifies the regression model as predicting **y** from **x**, **c1**, and **c2**.

```
proc glm data=temp noprint outstat=regResults;
  by iteration;
  model y=x c1 c2;
run;quit;
```

9. The **regResults** data set contains more information than you want. It has Type I and Type III tests of all three regression parameters for each iteration. Read the data set into IML using a WHERE statement to read in only observations corresponding to Type III tests of the regression coefficient for **x**. Read in only the **prob** variable, which provides the *p*-value for the test of the regression coefficient.

```
proc iml;
  use regResults;
  read all var {prob}
    where(_SOURCE_='X' & _TYPE_='SS3') into prob;
  close regResults;
```

10. Finally, calculate the proportion of elements in **prob** that are less than .05 and output the result.

```
significant=prob < .05;
power=significant[:,];
print power;
quit;
```

power
0.806

You see that the empirical estimate for the power to detect a significant partial regression coefficient for *X* is .806.

 Performing this power analysis in the POWER procedure would yield an analytical estimate of power at .799. The empirical estimate of power would converge to this number for larger sample sizes (for example, one million iterations).

End of Demonstration



Calling R from SAS

Use the randomForest package in R. Send the **birth** data set to R. Use the randomForest() function to create a predictive model, and return the results to SAS.

1. Invoke SAS/IML and send the **birth** data set in the **Work** library to R. Name the data frame **birth** as well.

```
proc iml;
  call ExportDataSetToR("work.birth","birth");
```

2. Write your R code between the SUBMIT and ENDSUBMIT statements. Use the randomForest package in R and the randomForest() function to estimate a model with **BWT** as the dependent variable and **Smoke**, **HT**, **LWT**, and **PTL** as independent variables. Use the SUMMARY statement to print the details of the analysis to the console. Finally, create a data frame with the actual and predicted values, given the model, and name the variables **Actual** and **Predicted**.

```
submit / r;
  library(randomForest)
  rf = randomForest(BWT ~ SMOKE + HT + LWT + PTL,
    data=birth,ntree=200,importance=TRUE)
  summary(rf)
  actual = birth$BWT
  pred = predict(rf,data=birth)
  actual.pred = cbind(actual,pred)
  colnames(actual.pred) <- c("Actual","Predicted")
endsubmit;
```

	Length	Class	Mode
call	5	-none-	call
type	1	-none-	character
predicted	189	-none-	numeric
mse	200	-none-	numeric
rsq	200	-none-	numeric
oob.times	189	-none-	numeric
importance	8	-none-	numeric
importanceSD	4	-none-	numeric
localImportance	0	-none-	NULL
proximity	0	-none-	NULL
ntree	1	-none-	numeric
mtry	1	-none-	numeric
forest	11	-none-	list
coefs	0	-none-	NULL
y	189	-none-	numeric
test	0	-none-	NULL
inbag	0	-none-	NULL
terms	3	terms	call



The output generated in the R console was printed in the SAS Results page.

3. Return the data frame to a SAS data set with the name **Rdata**.

```
call ImportDataSetFromR("Rdata","actual.pred");  
quit;
```

-  If you are running SAS Studio in client-server mode, you do **not** have access to the **Work** library on a point-and-click basis. You must use the PRINT procedure to view the results.

End of Demonstration

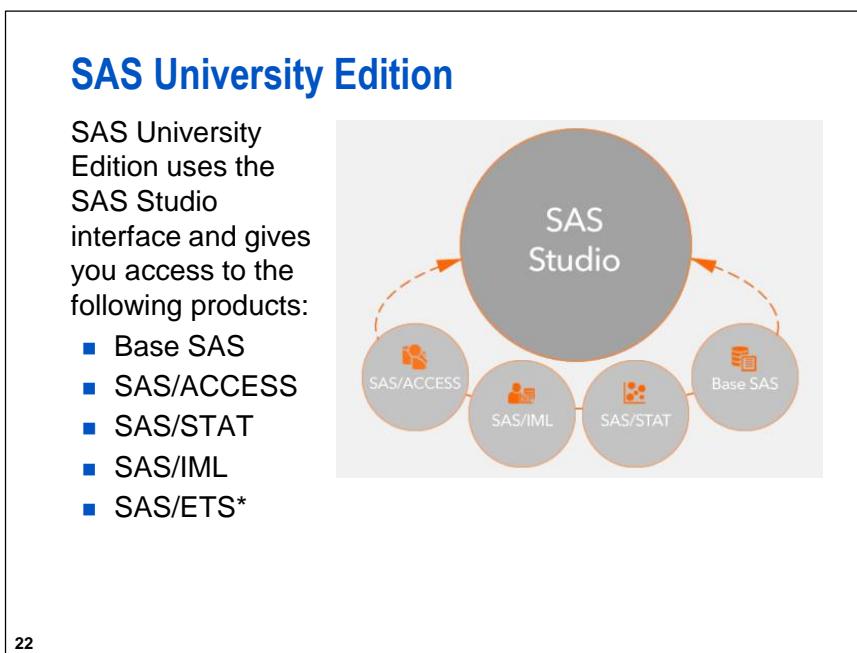


SAS University Edition is free SAS software that can be used for teaching and learning statistics and quantitative methods. It is designed for those who want easy access to statistical software. This includes the following groups:

- teachers and professors
- academic researchers
- students
- independent learners

21

To download SAS University, go to http://www.sas.com/en_us/software/university-edition.html.



22

Base SAS: The foundation for all SAS software. It provides a highly flexible, highly extensible, fourth-generation programming language and a rich library of programming procedures.

SAS/ACCESS: Seamlessly connect with your data no matter where it resides or how it is saved. SAS/ACCESS provides tools to easily access external data.

SAS/STAT: Provides a wide variety of statistical methods and techniques.

SAS/IML: A matrix programming language for more specialized analyses.

SAS/ETS: A suite of time series forecasting procedures. SAS University Edition offers only the TIMEDATA, TIMESERIES, ARIMA, ESM, UCM, and TIMEID procedures.

1.01 Multiple Choice Poll

Choose the correct statement.

- a. SAS has a command line interpreter.
- b. SAS is case sensitive,
- c. SAS Studio and SAS University Edition are synonymous.
- d. SAS applies procedures to the data table for analysis.

23

Procedures, Statements, and Options

- SAS procedures carry out all types of analyses.
- SAS statements contain keywords that request SAS to perform an operation or give information to the system.
 - A statement always ends with a semicolon.
- SAS options are additional arguments that are supplied to a procedure. They can be used to
 - generate additional output (results and plots)
 - save output to a SAS data table
 - alter the analytical method.

```
proc mcmc data=work.birth
  outpost=birthout propcov=quanew nmc=200000
  plots=all seed=27513;
  ...
  model low ~ binary(p);
run;
```

25

SAS Documentation

SAS documentation is an online resource that provides a consistent and detailed description of the features of SAS.

- procedures
- statements
- options
- analytical methods
- SAS syntax



- ☞ The SAS online documentation is at
<http://support.sas.com/documentation>.



Navigating the Online Documentation

1. Go to the SAS online documentation at <http://support.sas.com/documentation/>.

You can search SAS syntax using the Syntax Shortcuts or enter a search item in the query box.

2. Click **SAS Analytics Products 14.1**. This is the most recent version of SAS Analytics, a software suite developed by SAS to accomplish different tasks.

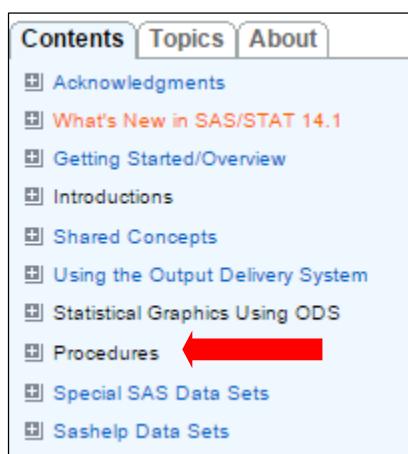
The following are different licenses of SAS software.

SAS/ETS	provides extensive facilities for analyzing time series and performing financial analysis.
SAS Enterprise Miner	streamlines the data mining process and creates both predictive and descriptive models based on vast amounts of data.
SAS Factory Miner	is a web-based tool that enables you to build models for every segment of your business using a variety of machine learning techniques.
SAS/IML	provides a matrix language to perform simple or complex matrix operations.
SAS/IML Studio	contains a highly flexible programming language that can run SAS/STAT and SAS/IML and display graphics and data tables.

SAS/OR	integrates essential optimization, scheduling, simulation, and related modeling solution capabilities in an adaptable environment. It enables companies to optimize business processes and management challenges.
SAS/QC	provides a comprehensive set of tools for statistical quality improvement and design of experiments.
SAS/STAT	enables the use of high-performance modeling tools for massive data.
SAS Text Miner	enables you to extract information from a collection of text documents and uncover themes and concepts previously concealed.

 For this course, you use SAS/STAT and SAS/IML extensively.

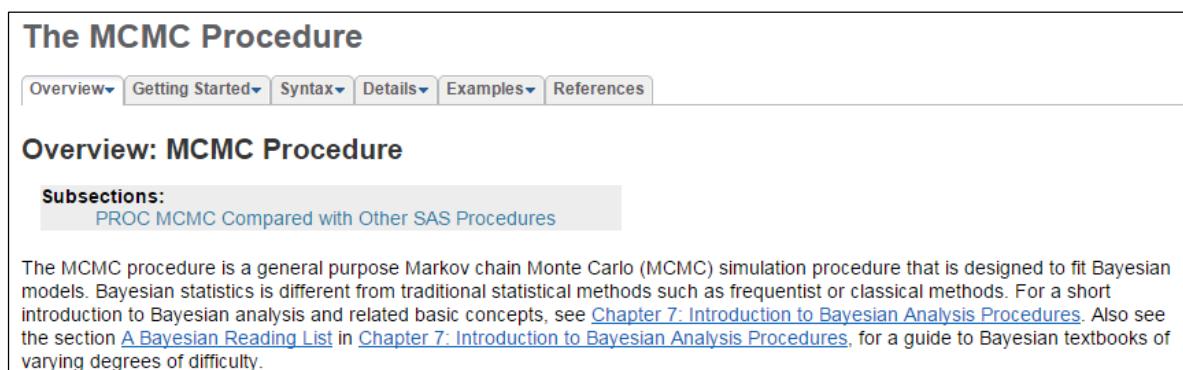
3. Select **What's New in SAS/STAT 14.1**.
4. Select **Procedures** in the Contents pane.



The screenshot shows the SAS/STAT 14.1 Contents pane. At the top, there are three tabs: 'Contents' (selected), 'Topics' (highlighted in orange), and 'About'. Below the tabs is a list of topics. A red arrow points to the 'Procedures' link, which is located under the 'Statistical Graphics Using ODS' section. Other visible links include 'Acknowledgments', 'What's New in SAS/STAT 14.1' (highlighted in orange), 'Getting Started/Overview', 'Introductions', 'Shared Concepts', 'Using the Output Delivery System', 'Statistical Graphics Using ODS', 'Special SAS Data Sets', and 'Sashelp Data Sets'.

This category lists all the statistical procedures that are available in SAS/STAT. You can also click the **Topics** tab to view procedures according to analysis.

5. Select **The MCMC Procedure** and then select the **Overview** tab.



The screenshot shows the 'The MCMC Procedure' page. At the top, there is a navigation bar with tabs: 'Overview' (selected), 'Getting Started', 'Syntax', 'Details', 'Examples', and 'References'. Below the navigation bar is a section titled 'Overview: MCMC Procedure'. Underneath this section, there is a 'Subsections' box containing a link to 'PROC MCMC Compared with Other SAS Procedures'. The main content area describes the MCMC procedure as a general purpose Markov chain Monte Carlo (MCMC) simulation procedure designed for Bayesian models. It compares Bayesian statistics with frequentist methods and provides links to Chapter 7: Introduction to Bayesian Analysis Procedures and a Bayesian Reading List.

The Overview tab discusses the procedure and the analyses that it conducts.

6. Click the Syntax tab.

The MCMC Procedure

Overview ▾ Getting Started ▾ Syntax ▾ Details ▾ Examples ▾ References

Syntax: MCMC Procedure

The following statements are available in the MCMC procedure. Items within < > are optional.

PROC MCMC <options>;
ARRAY arrayname [dimensions] <\$> <variables-and-constants>;
BEGINCNST/ENDCNST ;
BEGINNODATA/ENDNODATA ;
BY variables;
MODEL variable ~distribution <options>;
PARMS parameter <=> number </ options>;
PREDIST <'label'> OUTPRED=SAS-data-set <options>;
PRIOR/HYPERTPRIOR parameter ~distribution;
Programming statements ;
RANDOM random-effects-specification </ options>;
UDS subroutine-name (subroutine-argument-list);

The Syntax tab lists all the statements of the procedure and the subsequent code that the user needs to provide. The <> brackets denotes optional syntax. You can select any statement to view options that are available to the user.

7. Select **PROC MCMC** to view the statement options.

Table 73.1: PROC MCMC Statement Options

Option	Description
Basic options	
DATA=	Names the input data set
OUTPOST=	Names the output data set for posterior samples of parameters
Debugging output	
LIST	Displays the model program and variables
LISTCODE	Displays the compiled model program
TRACE	Displays detailed model execution messages
Frequently used MCMC options	
ALG=	Specifies the default sampling algorithm
MAXTUNE=	Specifies the maximum number of tuning loops
MINTUNE=	Specifies the minimum number of tuning loops
NBI=	Specifies the number of burn-in iterations
NMC=	Specifies the number of MCMC iterations, excluding the burn-in iterations
NTHREADS=	Specifies the number of threads to use
NTU=	Specifies the number of tuning iterations
PROPCOV=	Controls options for constructing the initial proposal covariance matrix
SEED=	Specifies the random seed for simulation
THIN=	Specifies the thinning rate

8. Select the **OUTPOST=** option to view a more detailed description.

OUTPOST=SAS-data-set
 specifies an output data set that contains the posterior samples of all model parameters, the iteration numbers (variable name ITERATION), the log of the posterior density (LOGPOST), the log of the prior density (LOGPRIOR), the log of the hyperprior density (LOGHYPER), if the [HYPER](#) statement is used, and the log likelihood (LOGLIKE). Any secondary parameters (assigned using the operator "=") listed in the [MONITOR=](#) option are saved to this data set. By default, no OUTPOST= data set is created.

9. There are also opportunities to learn about statistics in the documentation. Select **Metropolis** and **Metropolis-Hastings Algorithms** under the PROPDIST= option.

1. Set $t = 0$. Choose a starting point θ^0 . This can be an arbitrary point as long as $f(\theta^0|\mathbf{y}) > 0$.
2. Generate a new sample, θ_{new} , by using the proposal distribution $q(\cdot|\theta^t)$.
3. Calculate the following quantity:
$$r = \min \left\{ \frac{f(\theta_{\text{new}}|\mathbf{y})}{f(\theta^t|\mathbf{y})}, 1 \right\}$$
4. Sample u from the uniform distribution $U(0, 1)$.
5. Set $\theta^{t+1} = \theta_{\text{new}}$ if $u < r$; otherwise set $\theta^{t+1} = \theta^t$.
6. Set $t = t + 1$. If $t < T$, the number of desired samples, return to step 2. Otherwise, stop.

10. Go back one page and click the **Examples** tab. Then select **Logistic Regression Model with a Diffuse Prior**.

```
title 'Logistic Regression Model with a Diffuse Prior';
data beetles;
  input n y x @@;
  datalines;
  6 0 25.7 8 2 35.9 5 2 32.9 7 7 50.4 6 0 28.3
  7 2 32.3 5 1 33.2 8 3 40.9 6 0 36.5 6 1 36.5
  6 6 49.6 6 3 39.8 6 4 43.6 6 1 34.1 7 1 37.4
  8 2 35.2 6 6 51.3 5 3 42.5 7 0 31.3 3 2 40.6
;;

```

You can model the data points y_i with a binomial distribution,

$$y_i|p_i \sim \text{binomial}(n_i, p_i)$$

where p_i is the success probability and links to the regression covariate x_i through a logit transformation:

$$\text{logit}(p_i) = \log\left(\frac{p_i}{1 - p_i}\right) = \alpha + \beta x_i$$

The priors on α and β are both diffuse normal:

$$\begin{aligned} \alpha &\sim \text{normal}(0, \text{var} = 10000) \\ \beta &\sim \text{normal}(0, \text{var} = 10000) \end{aligned}$$

These statements fit a logistic regression with PROC MCMC:

```
ods graphics on;
proc mcmc data=beetles ntu=1000 nmc=20000 propcov=quanew
           diag=(mcse ess) outpost=beetleout seed=246810;
  ods select PostSumInt mcse ess TADpanel;
  parms (alpha beta) 0;
  prior alpha beta ~ normal(0, var = 10000);
  p = logistic(alpha + beta*x);
  model y ~ binomial(n,p);
run;
```

The documentation provides SAS syntax examples, which might be more helpful when you become comfortable with SAS syntax.

End of Demonstration

1.02 Quiz

Navigate to the SAS online documentation and find the LOGISTIC procedure. This procedure is used to conduct logistic regression when the response is binary. What does the DIFF option do in the LSMEANS statement?

28

SAS Training

Another helpful online tool is <http://support.sas.com/training/>.

The screenshot shows the SAS Training website. At the top, there's a navigation bar with links for support.sas.com, Knowledge Base, Support, Training & Books, Happenings, Store, and Support Communities. Below this is a header for 'TRAINING & BOOKS / TRAINING'. On the left, a sidebar lists categories like Books, Training, Certification, and SAS OnDemand. The main content area features a banner with the text 'Get certified in SAS®' and 'Stand out from the crowd with a SAS® Global Certification.' It includes a 'Get certified >' button and several 'Starting Points' links such as 'Log in to My Training', 'Get started with SAS', 'Manage your registrations', 'Train a group and save.', and 'View Top 10 course list.'

30

Free Tutorials

Visit <http://support.sas.com/training/tutorial/index.html>.

New to SAS	Visual Analytics	Analytics	Foundation Tools	Data Management	Business Intelligence	Administration	Solutions
Getting Started	Performing Statistical Analyses						
What is SAS?	Summary Statistics Using SAS Studio						
Getting Started with SAS Studio	Distribution Analysis Using SAS Studio						
Using SAS	One-Way Frequency Analysis Using SAS Studio						
Working in SAS Studio	Table Analysis Using SAS Studio						
Getting Started with SAS Enterprise Guide	Correlation Analysis Using SAS Studio						
Getting Started with SAS Windowing Environment	One-Sample t Tests Using SAS Studio						
Writing a Basic SAS Program	Paired-Sample t Tests Using SAS Studio						
Accessing Data	Two-Sample t Tests Using SAS Studio						
Accessing Data in SAS Libraries	One-Way ANOVA Using SAS Studio						
Creating a SAS Table from a CSV File	N-Way ANOVA Using SAS Studio						
Reading and Generating CSV Files Using Snippets in SAS Studio	Nonparametric One-Way ANOVA Using SAS Studio						
Using the Import Data Utility in SAS Studio	Simple Linear Regression Using SAS Studio						

31

The Analytics tab has many free videos that demonstrate statistical tasks.

Evolution of the Course

- Part 1: The Basics**
- Data Entry
 - Modifying Data Sets
 - Statistical Graphics

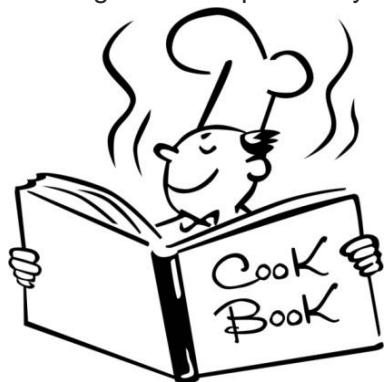
- Part 2: Statistical Procedures**
- Data Exploration
 - Inferential Modeling

- Part 3: Matrix Land**
- Interactive Matrix Language
 - Simulation
 - Integrating R and SAS

32

The Instructor's Goals

- In this course, the instructor provides you with the “R to SAS Cookbook.”
- You obtain the tools that enable you to learn additional SAS programming skills independently.



33

1.2 SAS Programming (Self-Study)

Objectives

- Compare SAS and R in a general way.
- List the components of a SAS program.
- Define SAS syntax rules.
- Use comments to document a program.
- Diagnose and correct a program that contains errors.

35

SAS versus R in General

SAS	R
Script compiler	Command line interpreter
Primarily driven by the data table and procedures	Object oriented
Not case sensitive	Case sensitive

SAS does have the flexibility to interact with objects.
This course focuses on procedural methods.

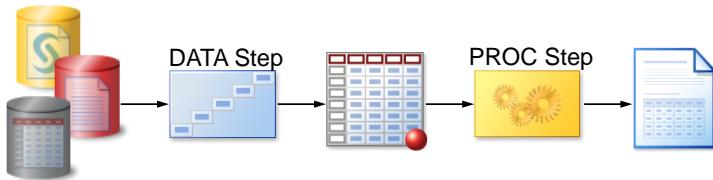
36

SAS does not have a command line. Code must be run in order to return results.

R is an object-oriented programming language. Results of a function are stored in an object and desired results are pulled from the object as needed. SAS revolves around the data table and uses procedures to create and print output. Results can be saved to a new data table.

SAS Programs

A SAS *program* is a sequence of one or more steps.



- *DATA steps* typically create SAS data sets.
- *PROC steps* typically process SAS data sets to generate reports and graphs, and to manage data.

37

A SAS program is a sequence of steps. There are only two types of steps in SAS: DATA and PROC steps.

- DATA steps read from an input source and create a SAS data set.
- PROC steps read and process a SAS data set, often generating an output report. Think of a PROC step as a function in R.

Step Boundaries

SAS steps begin with either of the following statements:

- a DATA statement
- a PROC statement

SAS detects the end of a step when it encounters one of the following statements:

- a RUN statement (for most steps)
- a QUIT statement (for some procedures)

38

Every step has a beginning and ending boundary. Most steps end with a RUN statement. Some PROC steps end with a QUIT statement. Think of the RUN statement as the right parentheses of an R function.

SAS Program Steps

A *step* is a sequence of SAS statements. This program has a DATA step and a PROC step.

```
data work.newemps;
  infile "&path\newemps.csv" dlm=',';
  input First $ Last $ Title $ Salary;
run;

proc print data=work.newemps;
run;
```

- R uses parentheses to begin and end a function.



The screenshot shows the RStudio interface with the following code in the script pane:

```
work.newemps = read.csv("C:/Users/jobake/Desktop/work.newemps.csv")
print(work.newemps)
```

39

This program contains the following statements:

- DATA
- INFILE
- INPUT
- RUN
- PROC

SAS Syntax Rules: Statements

SAS statements

- begin with an ***identifying keyword***
- always end with a ***semicolon***.

```
proc print data=work.newsalesemps;
run;

proc means data=work.newsalesemps;
  var Salary;
run;
```

40

SAS statements usually begin with a keyword, and *always* end with a semicolon. Keywords identify the type of statement, and semicolons end the statement.

1.03 Short Answer Poll

How many statements are contained in this DATA step?

```
data work.newsalesemps;
  length First_Name $ 12
    Last_Name $ 18 Job_Title $ 25;
  infile "&path\newemps.csv" dlm=',';
  input First_Name $ Last_Name $
    Job_Title $ Salary;
run;
```

41

Syntax Errors

A *syntax error* is an error in the spelling or grammar of a SAS statement. SAS finds syntax errors as it compiles each SAS statement, before execution begins.

Examples of syntax errors:

- misspelled keywords
- unmatched quotation marks
- invalid options
- missing semicolons

```
proc print data=work.newsalesemp  
run;
```

43

Syntax Errors

- The Enhanced Editor in SAS uses the color red to indicate a potential error in your SAS code.
- The RUN statement is bold and blue if all statements correctly end with a semicolon.

```
daat work.newsalesemps;  
length First_Name $ 12  
      Last_Name $ 18 Job_Title $ 25;  
infile "&path\newemps.csv" dlm=':';  
input First_Name $ Last_Name $  
      Job_Title $ Salary;  
run;  
  
proc print data=work.newsalesemps  
run;  
  
proc means data=work.newsalesemps average min;  
var Salary;  
run;
```

44

The Enhanced Editor uses the color red to indicate a potential error in your SAS code. Notice that the misspelled word **D-A-A-T** is displayed in red. This misspelling affects other statements following it because those statements are only permitted in a DATA step, and this is not recognized as such.

The RUN statement in the PROC PRINT step is not the correct font or color, and neither is the word “average” in the PROC MEANS statement.

Code can contain incorrect keywords. For example, “average” is not recognized by the PROC MEANS statement. MEAN is the correct word.

Error messages are written to the SAS log to describe syntax errors.

These are a few of the options used in the demonstration.

- OUTPUT= names the output data set for posterior samples of parameters.
- NMC= specifies the number of MCMC iterations, excluding the burn-in iterations.
- PLOTS= specifies which plots to print.
- SEED= specifies the random number seed (default is 0) to reproduce the analysis.
- PROPCOV= specifies the method used in constructing the initial covariance matrix for the Metropolis-Hastings algorithm. Specifically, it specifies the convergence method to use for optimization.

SAS Comments

This program contains four comments.

```
*-----*
| This program creates and uses the |
| data set called work.newsalesemps. | ①
*-----*;
data work.newsalesemps;
  length First_Name $ 12 Last_Name $ 18
    Job_Title $ 25;
  infile "&path\newemps.csv" dlm=',';
  input First_Name $ Last_Name $
    Job_Title $ Salary /*numeric*/; ②
run;
/*
proc print data=work.newsalesemps; ③ /* comment */
run;
*/
proc means data=work.newsalesemps;
  *var Salary; ④
run;
```

45

This program contains four comments. The first comment describes the program. The second comment is within a statement. The third comment is commenting out a step. The fourth comment is commenting out a statement. R comments do not have an end. They simply comment out everything to the right of the # symbol. SAS comments are more functional.

To comment multiple lines simultaneously, highlight the lines. Hold down the Ctrl key and press /.

To uncomment, highlight the lines. Hold down the Ctrl and Shift keys and press /.

1.04 Multiple Choice Poll

Which statement is a SAS syntax requirement?

- a. Begin each statement in column one.
- b. Put only one statement on each line.
- c. Separate each step with a line space.
- d. End each statement with a semicolon.

46

1.3 Accessing Data in SAS Libraries

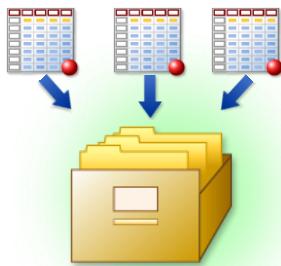
Objectives

- Discuss the use and creation of SAS libraries.
- Define the naming convention for SAS programs.
- Generate the course data.

49

SAS Libraries

SAS tables are stored in SAS libraries. A *SAS library* is a collection of SAS files that are referenced and stored as a unit.



Each file is a member of the library.

50

Temporary Library

Work is a temporary library where you can store and access SAS tables for the duration of the SAS session. It is the default library.



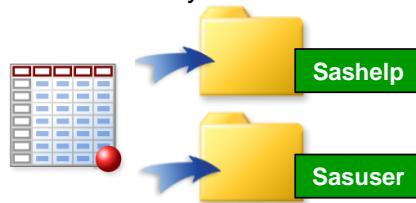
 SAS deletes the **Work** library and its contents when the SAS session ends.

51

Permanent Libraries

Sashelp is a permanent library that contains sample SAS tables that you can access during your SAS session.

Sasuser is a permanent library that you can use to store and access SAS tables in any SAS session.



52

User-Defined Libraries

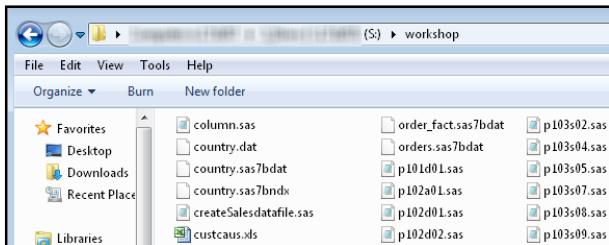
Users can create their own SAS libraries.

- A user-defined library is permanent. Tables are stored until the user deletes them.
- A user-defined library is implemented within the operating environment's file system.
- It is not automatically available in a SAS session.

53

Accessing a Permanent Library

Step 1 Identify the location of the library.



In this example, **s:\workshop**, a Microsoft Windows folder, is used as the SAS library.

54

You can use an existing folder or create a new one. After a folder is identified or created, the Windows operating system knows about the folder, but SAS does not. To use this folder as a SAS library, you must tell SAS about it. Sometimes this is referred to as making a connection between SAS and the folder.

Accessing a Permanent Library

Step 2 Use a SAS LIBNAME statement to associate the libref with the physical location of the library.



55

The concept of a SAS library is the same regardless of the operating environment, but libraries have different physical implementations depending on the environment. In UNIX and Windows, a library is a directory or folder. On a mainframe, it is an operating system file.

The path must be written in a style appropriate for the environment, and should include a full path. Examples are shown below. In class, you should open and submit libname.sas, because this program uses the location of your data.

Windows: libname perm 'S:\workshop';
UNIX: libname perm '~/workshop';
z/OS: libname perm 'userid.workshop.sasfiles';

LIBNAME Statement

The SAS LIBNAME statement is a *global* SAS statement.

```
libname SP4R "s :\workshop";  
LIBNAME libref "SAS-library" <options>;
```

- It is not required to be in a DATA step or PROC step.
- It does not require a RUN statement.
- It executes immediately.
- It remains in effect until changed or canceled, or until the session ends.

56

-  In the Microsoft Windows environment, an existing folder is used as a SAS library. The LIBNAME statement cannot create a new folder.
- In the UNIX environment, an existing directory is used as a SAS library. The LIBNAME statement cannot create a new directory.
- The LIBNAME statement makes this connection. You assign a short name called a libref or library reference name with the folder. The libref must be eight characters or less and begin with a letter or underscore followed by letters, underscores, and digits.
- In this example, you are associating the libref **SP4R** with the folder s:\workshop, which is the location of the course files.

Viewing the Log

Partial SAS Log

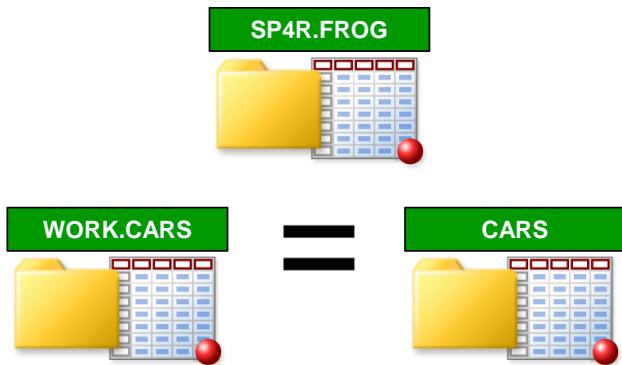
```
47 libname SP4R "s:\workshop";
NOTE: Libref SP4R was successfully assigned as follows:
      Engine: V9
      Physical Name: s:\workshop
```

57

Check the log after submitting a LIBNAME statement. Here you see that it executed successfully and assigned the libref **SP4R** with the physical folder s:\workshop.

Data Set Names

As a best practice, refer to **both** the library and the data set in DATA steps and PROC steps by using the convention *library.data-set-name*.



58

To access data in a permanent library, you must use the *library.data-set-name* convention. However, to access the temporary library **Work**, you do not need to use the library name. As a best practice, it is always encouraged to use the library name when you refer to a data set.

Course Data and Program Naming Convention

To create all the data used in this course, open and run the SAS file.

- ① course ID
- ② chapter #
- ③ type
 - a=activity
 - d=demo
 - e=exercise
 - s=solution
- ④ item #

sp4r01d02

SP4R, Chapter 1, Demo 2





Creating the Course Data and Library

SP4R01d02.sas

1. Create a macro variable that holds the path to the course folder.

```
%let path = s:\workshop;
```

2. Use a LIBNAME statement to create the **SP4R** library for the course. This library is linked to the s:\workshop folder. Any data created or altered is saved in this folder for later use.

```
libname sp4r "&path";
```



Refer to the s:\workshop path by referencing the macro variable as **&path**.

3. Create the course data by running the program SP4R01d03.sas in the s:\workshop file. The %INCLUDE statement simply runs the subsequent program.

```
%include "&path\SP4R01d03.sas";
```

4. Navigate to the **SP4R** library to view the course data.

End of Demonstration

1.4 Solutions

Solutions to Student Activities (Polls/Quizzes)

1.01 Multiple Choice Poll – Correct Answer

Choose the correct statement.

- a. SAS has a command line interpreter.
- b. SAS is case sensitive,
- c. SAS Studio and SAS University Edition are synonymous.
- d. SAS applies procedures to the data table for analysis.

24

1.02 Quiz – Correct Answer

Navigate to the SAS online documentation and find the LOGISTIC procedure. This procedure is used to conduct logistic regression when the response is binary. What does the DIFF option do in the LSMEANS statement?

The DIFF option requests the differences of LS-means.

29

1.03 Short Answer Poll – Correct Answer

How many statements are contained this DATA step?

```
data work.newsalesemps;
length First_Name $ 12
      Last_Name $ 18 Job_Title $ 25;
infile "&path\newemps.csv" dlm=',';
input First_Name $ Last_Name $
      Job_Title $ Salary;
run;
```

This DATA step has five statements.

42

1.04 Multiple Choice Poll – Correct Answer

Which statement is a SAS syntax requirement?

- a. Begin each statement in column one.
- b. Put only one statement on each line.
- c. Separate each step with a line space.
- d. End each statement with a semicolon.

47

Chapter 2 Importing and Reporting Data

2.1 The DATA Step and Manual Data Entry	2-3
Demonstration: Creating and Viewing the Data Table	2-6
2.2 Importing Data	2-9
Demonstration: Importing Data Using a DATA Step	2-13
Demonstration: Importing Data Using PROC IMPORT	2-14
Demonstration: Creating a SAS Data Set from Delimited Data by Hand	2-16
2.3 Reporting the Data.....	2-17
2.4 Enhanced Reporting.....	2-23
Demonstration: Labels, Formats, and Informats.....	2-29
Exercises	2-31
2.5 Solutions	2-33
Solutions to Exercises	2-33
Solutions to Student Activities (Polls/Quizzes)	2-38

2.1 The DATA Step and Manual Data Entry

Objectives

- Use the DATA step to create data tables manually.

3

Motivation

Create a data set by hand.

First Name	Last Name	Age	Height
Jordan	Bakerman	27	68



DATA Step



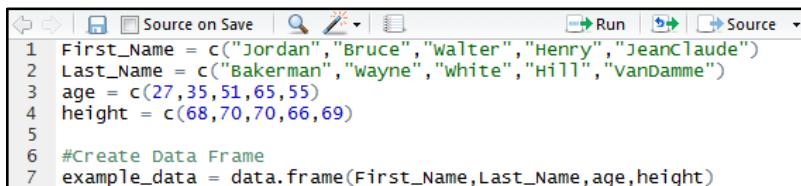
SAS data set



4

Duplicate the R Script

Create a SAS data table that is identical to the R data frame.



```

1 First_Name = c("Jordan", "Bruce", "Walter", "Henry", "JeanClaude")
2 Last_Name = c("Bakerman", "Wayne", "White", "Hill", "VanDamme")
3 age = c(27, 35, 51, 65, 55)
4 height = c(68, 70, 70, 66, 69)
5
6 #Create Data Frame
7 example_data = data.frame(First_Name,Last_Name,age,height)

```

	First_Name	Last_Name	age	height
1	Jordan	Bakerman	27	68
2	Bruce	Wayne	35	70
3	Walter	White	51	70
4	Henry	Hill	65	66
5	JeanClaude	VanDamme	55	69

5

The DATA Step

DATA step syntax generally uses five statements.

```

DATA new-data-set-name;
  LENGTH variable-a <$> # variable-b <$> #...;
  INPUT variable-a <$> variable-b <$>...;
  DATALINES;
  a1 b1 ... z1
  a2 b2 ... z2
  ...
  an bn ... zn
  ;
RUN;

```

6

The DATA step is used to create new data tables in SAS. Each DATA step begins with DATA followed by the name of the new data set and ends with a RUN statement.

Selected DATA step statements:

LENGTH specifies the length of each variable within the statement. The default length for all variables is eight bytes regardless of type. Variables not specified in the LENGTH statement are restricted to eight bytes. The specified length does not need to be exact. An upper bound can suffice. However, too large of an upper bound allocates unnecessary storage to blank space.

INPUT specifies the variables being created in the new data table.

DATALINES denotes data following the statement.

 Character variables specified in the LENGTH and INPUT statements must be followed by the \$ symbol. However, the INPUT statement does not require the \$ symbol if the LENGTH statement is used.

 The <> symbol denotes optional SAS syntax.



Creating and Viewing the Data Table

SP4R02d01.sas

- Duplicate the R data frame from the course notes.

```
data sp4r.example_data;
length First_Name $ 25 Last_Name $ 25;
input First_Name $ Last_Name $ age height;
datalines;
Jordan Bakerman 27 68
Bruce Wayne 35 70
Walter White 51 70
Henry Hill 65 66
JeanClaude VanDamme 55 69
;
run;
```

Submit the DATA step to create a data table.

- Check the log to verify that the data set **sp4r.example_data** was created with five observations and four variables.

```
176 data example_data;
177 length First_Name $ 25 Last_Name $ 25;
178 input First_Name $ Last_Name $ age height;
179 datalines;

NOTE: The data set SP4R.EXAMPLE_DATA has 5 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time          0.00 seconds
      cpu time          0.00 seconds

185 ;run;
```

- Navigate to the **SP4R** library and select **Example_data**.

	First_Name	Last_Name	age	height
1	Jordan	Bakeman	27	68
2	Bruce	Wayne	35	70
3	Walter	White	51	70
4	Henry	Hill	65	66
5	JeanClaude	VanDamme	55	69

- Create an identical data table except enter more than one observation per line.

 The **@@** option at the end of the INPUT statement enables the DATA step to read in more than one observation per line.

```
data sp4r.example_data2;
length First_Name $ 25 Last_Name $ 25;
input First_Name $ Last_Name $ age height @@;
datalines;
```

```
Jordan Bakerman 27 68 Bruce Wayne 35 70 Walter White 51 70  
Henry Hill 65 66 JeanClaude VanDamme 55 69  
;  
run;
```

5. Navigate to the SP4R library and select Example_Data2.

	First_Name	Last_Name	age	height
1	Jordan	Bakerman	27	68
2	Bruce	Wayne	35	70
3	Walter	White	51	70
4	Henry	Hill	65	66
5	JeanClaude	VanDamme	55	69

End of Demonstration

2.01 Multiple Answer Poll

Which DATA step options and statements are missing to correctly read in the **Class** data? (Select all that apply.)

- a. the @@ option to read in more than one observation per line
- b. the \$ option to read in character data
- c. the semicolon after the data
- d. The data set name is case sensitive and should be **CLASS**.

```
data class;
    input grades
    datalines;
    B- A A+ C+ F- A- A B+ B+ B
run;
```

2.2 Importing Data

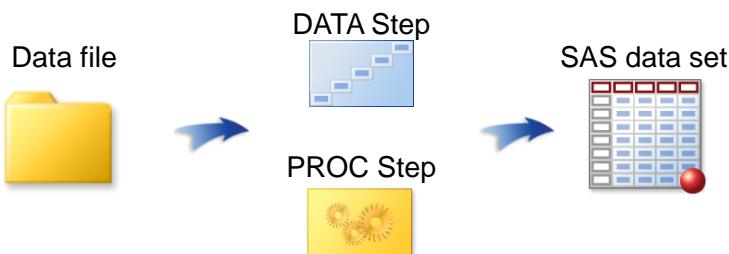
Objectives

- Import a delimited raw data file using a DATA step without column names.
- Import a delimited raw data file using PROC IMPORT with column names.
- Rename variables using the RENAME statement.

11

Motivation

Import a saved data file using either a DATA step or a PROC step.



12

Duplicate the R Script

- Import allnames.csv using a DATA step.
- Import baseball.csv using PROC IMPORT.

```
Source on Save Run Source
#Part I
#import data
allnames = read.csv("path/allnames.csv", header=FALSE)

#Change variable names
colnames(allnames) <- c("First_Name", "Last_Name", "age", "height")

#Part II
#import data with variable names
baseball = read.csv("path/baseball.csv")

#Change variable names
colnames(baseball) <- c("Name", "Team", "At_Bats", "Hits", "Home_Runs",
"Runs", "RBIs", "League", "Division",
"Position", "Errors")
```

13

Idea Exchange

What type of data files do you import in R?



14

Reading a Delimited Raw Data File

Replace the DATALINES statement with the INFILE statement to read a raw data file.

```
DATA output-data-set;
  LENGTH variable <$$> # variable <$$> #...;
  INFILE "data-file-path" DLM='delimiter'
    INPUT variable <$$> variable <$$>...;
RUN;
```

The INFILE statement identifies the raw data file to be read and requires the delimiter option, DLM, if the raw data file is separated by something other than a space.

```
infile "&path/example.csv" dlm=',';
      INFILE "data-file-path" DLM='delimiter';
```

15

The INFILE statement must come before the INPUT statement.

Some common delimiters are DLM=',' for .csv and DLM='09'x for tab-delimited files.

PROC IMPORT

Use PROC IMPORT to read in raw data files with column names.

```
PROC IMPORT OUT=data-set-name
  DATAFILE= "data-file-path"
  DBMS=identifier <REPLACE>;
  GETNAMES=<yes,no>;
  SHEET=<"sheet.name">
  DATAROW=<#>;
RUN;
```

- DBMS= specifies the type of data to import (for example, DBMS=csv for .csv files or xlsx for Excel workbooks).

16

PROC IMPORT is used similarly to the previous DATA step. However, PROC IMPORT is generally used for .csv files with the first row of the Excel file specifying the variable names.

Selected PROC IMPORT statements:

OUT names the SAS data table being created.
 DATAFILE specifies the path where the data set is located.
 DBMS specifies the type of data to import. For example, DBMS=csv is for .csv files and DBMS=tab is for tab-delimited files.
 GETNAMES equal to YES requests the first row of the input file to be used as variable names.
 SHEET specifies the name of the sheet to be imported in double quotation marks. The default is sheet 1.
 DATAROW specifies which row to begin reading in data. The default is DATAROW=2.

 The REPLACE option is used to write over existing SAS data tables with the same name.

Renaming Variables

Use the RENAME statement inside a DATA step to reproduce the colnames() function in R.

```
Source on Save | Run | Source
colnames(allnames) <- c("First_Name", "Last_Name", "age", "height")
```

```
data sp4r.allnames;
set sp4r.allnames;
rename VAR1=First_Name VAR2=Last_Name
          VAR3=Age VAR4=Height;
run;

DATA data-table-name-new;
SET data-table-name-old;
RENAME old-var-1 = new-var-1
       old-var-2 = new-var-2
       ...
       old-var-n = new-var-n;
RUN;
```

17

Selected DATA step statements:

SET specifies which data table to use as input for proceeding statements.
 RENAME used to rename variables. Notice that the semicolon is placed after the last renamed variable.
 If *data-table-name-new* = *data-table-name-old*, the data table is replaced with the new variable names.



Importing Data Using a DATA Step

SP4R02d02.sas

- The **allnames.csv** data set includes the five observations from **Example_Data** and 195 more observations for a total of 200. Use a DATA step to import the **allnames.csv** data set.

```
data sp4r.all_names;
length First_Name $ 25 Last_Name $ 25;
infile "&path\allnames.csv" dlm=',';
input First_Name $ Last_Name $ age height;
run;
```

- Check the log to verify that the data set **sp4r.all_names** was created with 200 observations and four variables.
- Navigate to the **SP4R** library and select **All_names**.

VIEWTABLE: Sp4r.All_names

	First_Name	Last_Name	age	height
1	Jordan	Bakeman	27	68
2	Bruce	Wayne	35	70
3	Walter	White	51	70
4	Henry	Hill	65	66
5	JeanClaude	VanDamme	55	69
6	Damion	Goodenow	33	81
7	Shannan	Moehrle	41	74
8	Zena	Seigfried	17	77
9	Cheree	Waldschmidt	41	77

End of Demonstration



Importing Data Using PROC IMPORT

SP4R02d02.sas

1. Import the file **baseball.csv**. This file refers to MLB player statistics from the 1986 season.

The first row of **baseball.csv** denotes the variable names.

The variables in the data set are listed below.

Name	player's name
Team	team at the end of the season
Natbat	times at bat
Nhits	number of hits
Nhome	number of home runs
Nruns	number of runs
Nrbi	number of RBIs
League	league (American, National)
Division	division (East, West)
Position	position on field
Nerror	number of errors

```
proc import out=sp4r.baseball
  datafile= "&path\baseball.csv" DBMS=CSV REPLACE;
  getnames=yes;
  datarow=2;
run;
```

Changing GETNAMES= to NO changes the variable names to **VAR1**, **VAR2**, **VAR3**, and so on.

2. Open the **baseball** data table to ensure that it was read in correctly. Then close the data table before you move to the next step.

You **must** close the data table **before** you write to it. If you use the subsequent DATA step when the data table is open, SAS throws an error.

3. Submit the DATA step to rename the variables **nAtBat**, **nHits**, **nHome**, **nRuns**, **nRBI**, and **nError**.

```
data sp4r.baseball;
  set sp4r.baseball;
  rename nAtBat = At_Bats
    nHits = Hits
    nHome = Home_Runs
    nRuns = Runs
    nRBI = RBIs
    nError = Errors;
run;
```

Navigate to the SP4R library and select **Baseball**.

Partial Table View

	Name	Team	At_Bats	Hits	Home_Runs	Runs	RBIs	League	Division	Position	Errors
1	Allanson, Andy	Cleveland	293	66	1	30	29	American	East	C	20
2	Ashby, Alan	Houston	315	81	7	24	38	National	West	C	10
3	Davis, Alan	Seattle	479	130	18	66	72	American	West	1B	14
4	Dawson, Andre	Montreal	496	141	20	65	78	National	East	RF	3
5	Galarraga, Andres	Montreal	321	87	10	39	42	National	East	1B	4
6	Griffin, Alfredo	Oakland	594	169	4	74	51	American	West	SS	25
7	Newman, Al	Montreal	185	37	1	23	8	National	East	2B	7
8	Salazar, Argenis	Kansas City	298	73	0	24	24	American	West	SS	9
9	Thomas, Andres	Atlanta	323	81	6	26	32	National	West	SS	19

End of Demonstration



Creating a SAS Data Set from Delimited Data by Hand

SP4R02d02.sas

In order to create a SAS data set from delimited data by hand, simply replace the data file path with the word DATALINES in the INFILE statement.

1. Create a data set by hand from * delimited data.

```
data sp4r.example_data3;
  length First_Name $ 25;
  infile datalines dlm='*';
  input First_Name $ Last_Name $ age height;
  datalines;
Jordan*Bakerman*27*68
Bruce*Wayne*35*70
Walter*White*51*70
Henry*Hill*65*66
Jean Claude*Van Damme*55*69
;run;
```

2. Navigate to the SP4R directory to view the data table.

	First_Name	Last_Name	age	height
1	Jordan	Bakeman	27	68
2	Bruce	Wayne	35	70
3	Walter	White	51	70
4	Henry	Hill	65	66
5	Jean Claude	Van Damm	55	69

End of Demonstration

2.3 Reporting the Data

Objectives

- Identify data table attributes with PROC CONTENTS.
- Identify variable levels with PROC SQL.
- Create a default PROC PRINT report.
- Select variables with a VAR statement.
- Select observations with a WHERE statement.
- Subset the data with a WHERE statement.

22

Motivation

Use a PROC step to report features of the imported data.



23

Duplicate the R Script

- Identify characteristics of the data set.
- Print conditional data.

```

#Print the first 6 rows
head(cars)

#Identify the names in the data set
names(cars)

#Identify the number of observations and variables
dim(cars)

#Identify the unique origins
levels(cars$origin)

#Print variables Type, origin, and MSRP where MSRP > $100,000
cars2 = data.frame(cars$type, cars$origin, cars$MSRP)
cars2[cars2$cars.MSRP>100000,]

#print variables Model, Type, and origin where Origin = Asia
cars2 = data.frame(cars$Model, cars>Type, cars$origin)
cars2[cars2$cars.origin=="Asia",]

#Print variables Make, origin, MSRP, and Type
#for Origin = Asia and MSRP > $75,000
cars2 = data.frame(cars$Make, cars$origin, cars$MSRP, cars>Type)
cars2[cars2$cars.origin=="Asia" & cars2$cars.MSRP>75000,]

```

24

The CONTENTS Procedure

PROC CONTENTS provides the same information as the functions dim() and names() in R.

```

proc contents data=sp4r.cars varnum;
run;

```

Partial Output

Data Set Name	WORK.CARS	Observations	428
Member Type	DATA	Variables	23

Variables in Creation Order					
#	Variable	Type	Len	Format	Label
1	mpg_quality	Char	6		
2	Make	Char	13		
3	Model	Char	40		

25



The VARNUM option is used to list the variables in order of appearance in the data table. The default is to list the variables in alphabetical order.

The PRINT Procedure

By default, PROC PRINT displays all observations and all variables.

- To reproduce the head() function in R, add the option (firstobs=1 obs=6) to the PRINT statement.

```
proc print data=sp4r.cars (firstobs=1 obs=6);
run;
```

Obs	mpg_quality	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice
1	Medium	Acura	MDX	SUV	Asia	All	\$36,945	\$33,337
2	Medium	Acura	RSX Type S 2dr	Sedan	Asia	Front	\$23,820	\$21,761
3	Medium	Acura	TSX 4dr	Sedan	Asia	Front	\$26,990	\$24,647
4	Medium	Acura	TL 4dr	Sedan	Asia	Front	\$33,195	\$30,299
5	Medium	Acura	3.5 RL 4dr	Sedan	Asia	Front	\$43,755	\$39,014
6	Medium	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	\$46,100	\$41,100

26

If you start from observation 1, you do not need FIRSTOBS=1.

The SQL Procedure

PROC SQL can be used to query the data table.

- To reproduce the levels() function in R, use PROC SQL with the SELECT UNIQUE statement.

```
proc sql;
  select unique origin from sp4r.cars;
quit;
```

```
PROC SQL;
  SELECT UNIQUE variable-name FROM data-table-name;
QUIT;
```

Origin
Asia
Europe
USA

27

Selected PROC SQL statements:

SELECT UNIQUE requests a list of the unique levels from the proceeding variable name.
 FROM refers SAS to the desired data table.

2.02 Multiple Choice Poll

Which SAS procedures are used to reproduce the R functions levels(), dim(), head(), and names()?

- a. PRINT, PRINT, PRINT, CONTENTS
- b. SQL, CONTENTS, PRINT, CONTENTS
- c. CONTENTS, PRINT, PRINT, SQL
- d. SQL, SQL, CONTENTS, PRINT

28

Additional PRINT Procedure Statements

Use the VAR and WHERE statements to alter the desired report.

```
PROC PRINT DATA=data-table <options>;
  VAR variable1 variable2 ...;
  WHERE conditional-expression;
  RUN;
```

- Examples of WHERE expressions

```
where salary>50000;
```

```
where gender='Male';
```

```
where upcase(gender)='MALE' ;
```

30

Selected PROC PRINT statements:

VAR specifies which variables should be printed.

WHERE specifies conditions to subset the printed output.

The WHERE statement requires the variable to match the observation exactly when you use character data. Use single quotation marks to specify the observation if it is character data. To avoid capitalization mistakes, use the UPCASE function to capitalize all the letters of the observation.

Comparison Operators

Symbol	Mnemonic	Definition
=	EQ	Equal to
~= ~=	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal
<=	LE	Less than or equal
	IN	Equal to one of a list

- Example of the IN operator

```
where country in ('US', 'CA');
```

31



SAS does not use the ! symbol to specify NOT in a comparison operator.



SAS does not use the == symbol to create a Boolean operator.

Logical Operator Priority

The operators can be written as symbols or mnemonics, and parentheses can be added to modify the order of evaluation.

Symbol	Mnemonic	Priority
^ ~	NOT	I
&	AND	II
	OR	III

The NOT operator modifies a condition by finding the complement of the specified criteria.

```
where Country not in ('US', 'CA');
```

32

2.03 Poll

The PROC step below prints the variables **grades**, **student**, and **year** from the **class** data set for all students with grades D or higher. (Assume that the data is clean and there are no + or – grades.)

- True
- False

```
proc print data=class;
  var grades student year;
  where upcase(grades)^='F';
run;
```

2.4 Enhanced Reporting

Objectives

- Use the LABEL statement to display descriptive column headings.
- Use the FORMAT procedure to create user-defined formats.
- Use the FORMAT statement to apply user-defined formats.
- Apply SAS formats with the FORMAT statement.

36

Motivation

Use labels and formats to alter the presentation of the data table or report.

FN	LN	Job	Bonus	Salary	Hire_Date
Tom	Zhou	SR	1	51500	12205
Joe	Wilson	SR	1	83975	6575
Alice	Elf	SM	0	94545	5575



First Name	Last Name	Job	Bonus	Salary	Hire Date
Tom	Zhou	Sales Rep	Yes	\$51,500	06/01/1993
Joe	Wilson	Sales Rep	Yes	\$83,975	01/01/1978
Alice	Elf	Sales Manager	No	\$94,545	04/07/1975

37

The LABEL Statement and Option

Use a LABEL statement and option to display descriptive column headings instead of variable names.

```
proc print data=sp4r.business label;
  label FN='First Name' LN='Last Name'
run;
```

LABEL variable-1='label-1' ... variable-n='label-n';

FN	LN
Tom	Zhou
Joe	Wilson
Alice	Elf

→

First Name	Last Name
Tom	Zhou
Joe	Wilson
Alice	Elf

38

Selected PROC PRINT statements:

VAR specifies which variables should be printed.
 LABEL specifies which variables should be labeled. The new label must be in quotation marks.

The LABEL option in the PROC PRINT statements must be included in addition to the LABEL statement.

What Is a Format?

A *format* is an instruction to write data values.

- A format changes the appearance of a variable's value in a report.
- The values stored in the data set are **not** changed.

SAS Date	01/10/1989
10866	10Jan1989

Numeric	5,950.35
5950.35	\$5,950.35

39

A *format* is an instruction that tells SAS how to display values in a report. A numeric value can be displayed with commas, and maybe with a dollar sign. A SAS date can be displayed in one of many calendar styles. Formats do not change the stored value.

SAS Formats

SAS formats have the following form:

`<$>format<w>.<d>`

\$	Indicates a character format.
<i>format</i>	Names the SAS format.
<i>w</i>	Specifies the total format width, including decimal places and special characters.
.	Is required syntax. Formats always contain a period (.) as part of the name.
<i>d</i>	Specifies the number of decimal places to display in numeric formats.

40

Formats begin with a \$ if it is a character format, followed by the name of the format, an optional width, a required dot delimiter. The format also contains an optional number of decimal places for numeric formats.

An extensive list of SAS formats can be found on the following page:

<http://support.sas.com/documentation/cdl/en/leforinforref/64790/HTML/default/viewer.htm#p0z62k899n6a7wn1r5in6q5253v1.htm>

SAS Format Examples

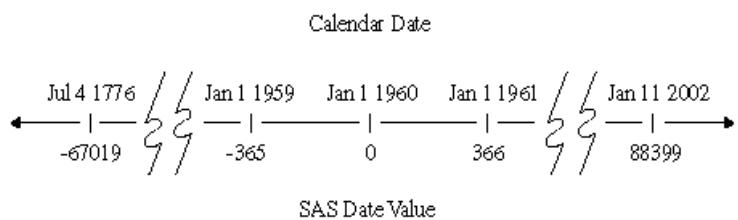
If the format width is not large enough to accommodate a numeric value, the displayed value is automatically adjusted to fit the width.

Format	Stored Value	Displayed Value
DOLLAR12.2	27134.5864	\$27,134.59
DOLLAR9.2	27134.5864	\$27134.59
DOLLAR8.2	27134.5864	27134.59
DOLLAR5.2	27134.5864	27135
DOLLAR4.2	27134.5864	27E3

41

SAS Date Format

SAS date formats display SAS date values in standard date forms. The value in the data table represents the number of days since January 1st, 1960, when you use a SAS date format.



42

SAS Date Format Examples

Format	Stored Value	Displayed Value
MMDDYY10.	0	01/01/1960
MMDDYY8.	0	01/01/60
MMDDYY6.	0	010160
DDMMYY10.	365	31/12/1960
DDMMYY8.	365	31/12/60
DDMMYY6.	365	311260

43

The Format Statement

Use the FORMAT statement to apply a SAS format.

```
proc print data=sp4r.business;
  format salary dollar8. hire_date mmddyy10.;
run;
```

FORMAT variable-name-1 format-name-1. ...;

Salary	Hire Date
51500	12205
83975	6575
94545	5575



Salary	Hire Date
\$51,500	06/01/1993
\$83,975	01/01/1978
\$94,545	04/07/1975

DOLLAR8.

MMDDYY10.

44

The FORMAT Procedure

Use PROC FORMAT to create a user-defined format for observation values.

```
proc format;
  value $jobformat 'SR'='Sales Rep'
    'SM'='Sales Manager';
  value bonusformat 0='No' 1='Yes';
run;
```

VALUE format-name range1='label1' ...;

- Each range can be a single value or a range of values.

45

Selected PROC FORMAT statement:

VALUE specifies the format name followed by the instructions for formatting values.
The FORMAT procedure might have multiple VALUE statements.

A format name can be a maximum of 32 characters in length. Character formats must begin with a dollar sign followed by a letter or underscore. Numeric formats must begin with a letter or underscore, cannot end in a number, cannot be given the name of a SAS format, and cannot include a period in the VALUE statement.

Labels can be a maximum of 32,767 characters in length and are enclosed in quotation marks.

-  LOW, HIGH, and OTHER are built-in SAS keywords that can be helpful when you format numeric data.

Applying the User-Defined Format

Use the FORMAT statement to apply the user-defined formats to the observations.

```
proc print data=sp4r.business;
  format job $jobformat. bonus bonusformat.;
run;
```

FORMAT variable-name-1 format-name-1. ...;

Job	Bonus
SR	1
SR	1
SM	0



Job	Bonus
Sales Rep	Yes
Sales Rep	Yes
Sales Manager	No

46

The format name for character data must start with a dollar sign. The name of the format must end with a period in the FORMAT statement. After the period is added, the format name becomes green.

DATA Step Labels and Formats

Use the LABEL statement or FORMAT statement (or both) in a DATA step to apply the labels and formats to

- a data table

VIEWTABLE: Work.Business					
	First Name	Last Name	Job	Bonus	Salary
1	Tom	Zhou	Sales Rep	Yes	\$51,500
2	Joe	Wilson	Sales Rep	Yes	\$83,975
3	Alice	Elf	Sales Manager	No	\$94,545
					06/01/1993
					01/01/1978
					04/07/1975

- analyses and reports that use the data table.

Job	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Sales Manager	1	33.33	1	33.33
Sales Rep	2	66.67	3	100.00

47

Applying labels and formats to a data table helps you avoid using the LABEL and FORMAT statements for each subsequent SAS procedure.



Labels, Formats, and Informats

SP4R02d03.sas

1. Use a DATA step and a SAS date informat to read in the following data table.

Name	Bday
Jill	01011960
Jack	05111988
Joe	08221975

The variable **bday** is already in the MMDDYY8. SAS format.

```
data employees;
  input name $ bday :mmddyy8. @@;
  datalines;
Jill 01011960 Jack 05111988 Joe 08221975
;
run;

proc print data=employees;
run;
```

Obs	name	bday
1	Jill	0
2	Jack	10358
3	Joe	5712

To read in data that is already in a SAS format, specify the format following the variable name in the INPUT statement. You **must** use the : (colon) symbol to precede the format name.

In addition, notice that SAS reads in the DATE format and stores the format according to the number of days since 1960.

2. Add a FORMAT statement to change the format of **bday** to MMDDYY10. and use a LABEL statement to change the names of the variables.

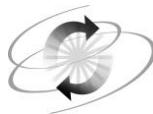
```
data employees;
  input name $ bday :mmddyy8. @@;
  format bday mmddyy10.;
  label name="First Name" bday="Birthday";
  datalines;
Jill 01011960 Jack 05111988 Joe 08221975
;
run;
```

3. Print the data table or navigate to the data table in the **Work** library to see the changes.

```
proc print data=employees label;
run;
```

Obs	First Name	Birthday
1	Jill	01/01/1960
2	Jack	05/11/1988
3	Joe	08/22/1975

End of Demonstration



Exercises

Your local animal shelter needs help with entering data into SAS. The information that they want to keep a record of is the **Name**, **Age**, **Weight**, and **Color** of each dog, and whether the dog likes **Cats**.

1. Creating a SAS Data Set by Hand

- Create a data set called **Shelter** in the **SP4R** library with the information in the table below (**SP4R02e01**).

Each line should correspond to a single observation.

Name	Age	Weight	Color	Cats
Pluto	3	25	Black	No
Lizzie	10	43	Tan	Yes
Pesci	10	38	Brindle	No

- Navigate to the **SP4R** directory to ensure that the **Shelter** data set was created correctly.
- Create another data set called **Shelter2**. This time, enter all of the data on a single line.
- Navigate to the **SP4R** directory to ensure that the **Shelter2** data set was created correctly.

2. Creating a Data Table with # Delimited Data

- Read the data below (**SP4R02e02**) into a DATA step with the name **Shelter3**. This is the same data from the **Shelter** data set exercise.

```
Pluto#3#Black#No
Lizzie#10#43#Tan#Yes
Pesci#10#38#Brindle#No
```

- Navigate to the data table in the **SP4R** directory.

3. Importing Data Using a DATA Step

- The 2010 Census population for each state is recorded in **state_pop.txt**. Locate and open **state_pop.txt**. How are the variables delimited?
- Import **state_pop.txt**. Use an INFILE statement inside a DATA step.
- Navigate to the data table in the **SP4R** directory.

4. Importing Data Using PROC IMPORT

- Use PROC IMPORT to import **state_pop.xlsx**. The data is located on the second sheet named **State_Pop_Data** and has no column names.
- Change the variable names by using a RENAME statement inside a DATA step.
- Navigate to the data table in the **SP4R** directory.

5. Labeling, Formatting, and Conditional Printing

Use the DATA step below (**SP4R02e05**), which generates the **Class** data table with 20 observations and four variables to complete the exercises.

```

data sp4r.class;
input student $ country $ grade bd @@;
datalines;
John Spain 95 12000 Mary Spain 82 12121 Alison France 98 12026
Nadine Spain 77 12222 Josh Italy 61 12095 James France 45 12301
William France 92 12284 Susan Italy 95 12079
Charlie France 88 12234 Alice Italy 89 12014 Robert Italy 92 12025
Emily Spain 87 12148 Arthur Italy 99 12052 Nancy France 70 12238
Kristin France 65 12084 Sara Italy 49 12322 Ashley Spain 96 12299
Aaron France 95 12052 Sean France 87 12254 Phil Italy 86 12036
;
run;

```

- a. Use PROC FORMAT to create a format for the **grade** variable.

Grade	GradeFormat
0-59	F
60-69	D
70-79	C
80-89	B
90-100	A

- b. Use the DATA step above to read in the **Class** data set. In the DATA step, label the variable **bd** as “Birthday” and apply the GradeFormat created in part a. In addition, use the SAS format WORDDATE for the **bd** variable.
- c. Print the **Class** data table. (Remember to use the LABEL option in the PRINT statement.)
- d. Use PROC SQL to print the unique levels of the **country** variable.
- e. Conditionally print the variable **student**, **country**, and **grade** for people with a grade above 79 and from France only.

End of Exercises

2.5 Solutions

Solutions to Exercises

Your local animal shelter needs help with entering data into SAS. The information that they want to keep a record of is the **Name**, **Age**, **Weight**, and **Color** of each dog, and whether the dog likes **Cats**.

1. Creating a SAS Data Set by Hand

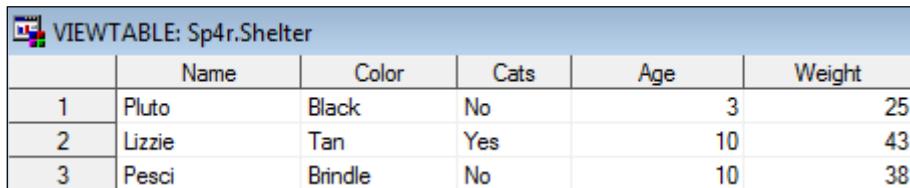
- Create a data set called **Shelter** in the **SP4R** library with the information in the table below (**SP4R02e01**).

Each line should correspond to a single observation.

Name	Age	Weight	Color	Cats
Pluto	3	25	Black	No
Lizzie	10	43	Tan	Yes
Pesci	10	38	Brindle	No

```
data sp4r.shelter;
length Name $ 25 Color $ 10 Cats $ 3;
input Name $ Age Weight Color $ Cats $;
datalines;
Pluto 3 25 Black No
Lizzie 10 43 Tan Yes
Pesci 10 38 Brindle No
;
run;
```

- Navigate to the **SP4R** directory to ensure that the **Shelter** data set was created correctly.



The screenshot shows a SAS ViewTable titled "VIEWTABLE: Sp4r.Shelter". The table has columns: #, Name, Color, Cats, Age, and Weight. There are three rows of data:

#	Name	Color	Cats	Age	Weight
1	Pluto	Black	No	3	25
2	Lizzie	Tan	Yes	10	43
3	Pesci	Brindle	No	10	38

- Create another data set called **Shelter2**. This time, enter all of the data on a single line.

```
data sp4r.shelter2;
length Name $ 25 Color $ 10 Cats $ 3;
input Name $ Age Weight Color $ Cats $ @@;
datalines;
Pluto 3 25 Black No Lizzie 10 43 Tan Yes Pesci 10 38 Brindle No
;
run;
```

- d. Navigate to the **SP4R** directory to ensure that the **Shelter2** data set was created correctly.

	Name	Color	Cats	Age	Weight
1	Pluto	Black	No	3	25
2	Lizzie	Tan	Yes	10	43
3	Pesci	Brindle	No	10	38

2. Creating a Data Table with # Delimited Data

- a. Read the data below (**SP4R02e02**) into a DATA step with the name **Shelter3**. This is the same data from the **Shelter** data set exercise.

```
data sp4r.shelter3;
  infile datalines dlm='#';
  input Name $ Age Weight Color $ Cats $;
  datalines;
  Pluto#3#25#Black#No
  Lizzie#10#43#Tan#Yes
  Pesci#10#38#Brindle#No
;run;
```

- b. Navigate to the data table in the **SP4R** directory.

	Name	Age	Weight	Color	Cats
1	Pluto	3	25	Black	No
2	Lizzie	10	43	Tan	Yes
3	Pesci	10	38	Brindle	No

3. Importing Data Using a DATA Step

- a. The 2010 Census population for each state is recorded in **state_pop.txt**. Locate and open **state_pop.txt**. How are the variables delimited? **The variables are tab delimited.**
- b. Import **state_pop.txt**. Use an INFILE statement inside a DATA step.

```
data sp4r.state_pop;
  length State $ 25;
  infile "&path\state_pop.txt" dlm='09'x;
  input State $ Population;
run;
```

- c. Navigate to the data table in the **SP4R** directory.

	State	Population
1	Alabama	4779736
2	Alaska	710231
3	Arizona	6392017
4	Arkansas	2915918
5	California	37253956
6	Colorado	5029196
7	Connecticut	3574097
8	Delaware	897934
9	Florida	18801310

4. Importing Data Using PROC IMPORT

- a. Use PROC IMPORT to import **state_pop.xlsx**. The data is located in the second sheet named **State_Pop_Data** and has no column names.

```
proc import out = state_pop2
  datafile = "&path\state_pop.xlsx"
  dbms = xlsx REPLACE;
  getnames = NO;
  sheet = "State_Pop_Data";
  datarow = 1;
run;
```

- b. Change the variable names by using a RENAME statement inside a DATA step.

```
data sp4r.state_pop2;
  set sp4r.state_pop2;
  rename VAR1 = State VAR2 = Population;
run;
```

- c. Navigate to the data table in the SP4R directory.

	State	Population
1	Alabama	4779736
2	Alaska	710231
3	Arizona	6392017
4	Arkansas	2915918
5	California	37253956
6	Colorado	5029196
7	Connecticut	3574097
8	Delaware	897934
9	Florida	18801310

5. Labeling, Formatting, and Conditional Printing

Use the DATA step below (**SP4R02e05**), which generates the **Class** data table with 20 observations and four variables, to complete the exercises.

```
data sp4r.class;
  input student $ country $ grade bd @@;
```

```

datalines;
John Spain 95 12000 Mary Spain 82 12121 Alison France 98 12026
Nadine Spain 77 12222 Josh Italy 61 12095 James France 45 12301
William France 92 12284 Susan Italy 95 12079
Charlie France 88 12234 Alice Italy 89 12014 Robert Italy 92 12025
Emily Spain 87 12148 Arthur Italy 99 12052 Nancy France 70 12238
Kristin France 65 12084 Sara Italy 49 12322 Ashley Spain 96 12299
Aaron France 95 12052 Sean France 87 12254 Phil Italy 86 12036
;
run;

```

- a. Use PROC FORMAT to create a format for the **grade** variable.

Grade	GradeFormat
0-59	F
60-69	D
70-79	C
80-89	B
90-100	A

```

proc format;
  value gradesformat 0-59='F' 60-69='D' 70-79='C' 80-89='B'
    90-100='A';
run;

```

- b. Use the DATA step above to read in the **Class** data set. In the DATA step label the variable **bd** as “Birthday” and apply the GradeFormat created in part a. In addition, use the SAS format WORDDATE for the **bd** variable.

```

data sp4r.class;
  input student $ country $ grade bd @@;
  label bd='Birthday';
  format grade gradesformat. bd woddate.;
datalines;
John Spain 95 12000 Mary Spain 82 12121 Alison France 98 12026
Nadine Spain 77 12222 Josh Italy 61 12095 James France 45 12301
William France 92 12284 Susan Italy 95 12079
Charlie France 88 12234 Alice Italy 89 12014 Robert Italy 92 12025
Emily Spain 87 12148 Arthur Italy 99 12052 Nancy France 70 12238
Kristin France 65 12084 Sara Italy 49 12322 Ashley Spain 96 12299
Aaron France 95 12052 Sean France 87 12254 Phil Italy 86 12036
;
run;

```

- c. Print the **Class** data table. (Remember to use the LABEL option in the PRINT statement.)

```

proc print data= sp4r.class label;
run;

```

Obs	student	country	grade	Birthday

1	John	Spain	A	November 8, 1992
2	Mary	Spain	B	March 9, 1993
3	Alison	France	A	December 4, 1992
4	Nadine	Spain	C	June 18, 1993
5	Josh	Italy	D	February 11, 1993
6	James	France	F	September 5, 1993
7	William	France	A	August 19, 1993
8	Susan	Italy	A	January 26, 1993
9	Charlie	France	B	June 30, 1993
10	Alice	Italy	B	November 22, 1992
11	Robert	Italy	A	December 3, 1992
12	Emily	Spain	B	April 5, 1993
13	Arthur	Italy	A	December 30, 1992
14	Nancy	France	C	July 4, 1993
15	Kristin	France	D	January 31, 1993
16	Sara	Italy	F	September 26, 1993
17	Ashley	Spain	A	September 3, 1993
18	Aaron	France	A	December 30, 1992
19	Sean	France	B	July 20, 1993
20	Phil	Italy	B	December 14, 1992

- d. Use PROC SQL to print the unique levels of the **country** variable.

```
proc sql;
  select unique country from sp4r.class;
quit;
```

country
France
Italy
Spain

- e. Conditionally print the variable **student**, **country**, and **grade** for people with a grade above 79 and from France only.

```
proc print data= sp4r.class;
  var student country grade;
  where grade>79 and country='France';
run;
```

Obs	student	country	grade
3	Alison	France	A
7	William	France	A
9	Charlie	France	B
18	Aaron	France	A
19	Sean	France	B

End of Solutions

Solutions to Student Activities (Polls/Quizzes)

2.01 Multiple Answer Poll – Correct Answers

Which DATA step options and statements are missing to correctly read in the Class data??? (Select all that apply.)

- a. the @@ option to read in more than one observation per line
- b. the \$ option to read in character data
- c. the semicolon after the data
- d. The data set name is case sensitive and should be CLASS.

```
data class;
  input grades $ @@;
  datalines;
  B- A A+ C+ F- A- A B+ B+ B
;run;
```

9

2.02 Multiple Choice Poll – Correct Answer

Which SAS procedures are used to reproduce the R functions levels(), dim(), head(), and names()?

- a. PRINT, PRINT, PRINT, CONTENTS
- b. SQL, CONTENTS, PRINT, CONTENTS
- c. CONTENTS, PRINT, PRINT, SQL
- d. SQL, SQL, CONTENTS, PRINT

29

2.03 Poll – Correct Answer

The PROC step below prints the variables **grades**, **student**, and **year** from the **class** data set for all students with grades D or higher. (Assume that the data is clean and there are no + or – grades.)

- True
- False

```
proc print data=class;
  var grades student year;
  where upcase(grades)^='F';
run;
```


Chapter 3 Creating New Variables, Functions, and Data Tables

3.1	Creating New Variables	3-3
3.2	Creating and Using Functions	3-13
	Demonstration: Navigating to the SAS Online Functions Documentation.....	3-22
3.3	Subsetting and Concatenating Data Tables.....	3-29
	Exercises	3-39
3.4	Solutions	3-42
	Solutions to Exercises	3-42
	Solutions to Student Activities (Polls/Quizzes)	3-46

3.1 Creating New Variables

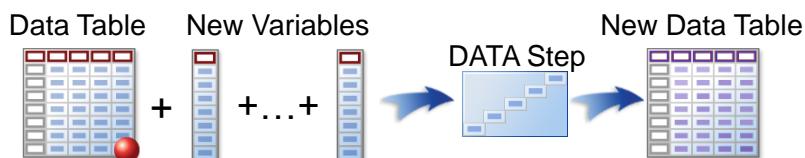
Objectives

- Create and add new variables to the data table.
- Use IF-THEN/ELSE statements to create new variables conditionally.

3

Motivation

Create and add new variables to the data table via the DATA step.



4

Duplicating the R Script

- Create and add new variables to the data frame.
- Use the if(), else if(), and else() functions.

```
#Create and add new variable to the data frame
cars$wheelbase_plus_length = cars$wheelbase + cars$length
cars(cars)

#create and add new variable conditionally
mpg_city_bonus = rep(NA,length(cars>Name))
for(i in 1:length(mpg_city_bonus)){
  if(cars$mpg_city[i]>=30){mpg_city_bonus[i]=2000}
  else if(cars$mpg_city[i]>=20){mpg_city_bonus[i]=1000}
  else {mpg_city_bonus[i]=0}
}

cars$mpg_city_bonus = mpg_city_bonus
```

5

Creating New Variables

Use a DATA step to create and add variables to the data table.

```
#Create and add new variable to the data frame
cars$wheelbase_plus_length = cars$wheelbase + cars$length
```



```
data sp4r.cars;
  set sp4r.cars;
  wheelbase_plus_length = wheelbase+length;
run;
```

```
DATA data-table-name-new;
  SET data-table-name-old;
  new-var-1 = expression-1;
  new-var-2 = expression-2;
  ...
  new-var-n = expression-3;
RUN;
```

6

Selected DATA step statements:

SET specifies which data table to use for proceeding statements.

- If *data-table-name-new* = *data-table-name-old*, the newly created variables are added to the existing data table.

Conditional Processing

Suppose car manufacturers receive an economic incentive for manufacturing cars with high highway miles per gallon.

Create a new variable to reflect the bonus that is received as outlined in the table below.

MPG Highway	Bonus
≥ 30	2,000
20-29	1,000
< 20	0

7

Assume that the Boston Red Sox received the highest television ratings of the 1986 MLB season followed by the Los Angeles Dodgers.

IF-THEN Statements

Use the IF-THEN statement to reproduce the if() function in R.

```
data sp4r.cars;
set sp4r.cars;
bonus=2000;
if mpg_highway<20 then bonus=0;
if mpg_highway>=20 and mpg_highway<30
    then bonus=1000;
run;
```

IF expression THEN statement;

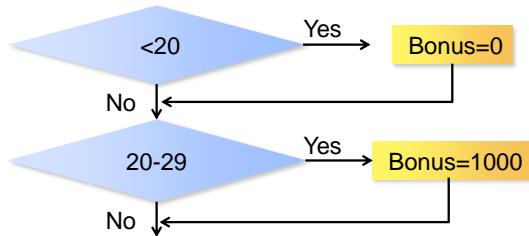
8



The **bonus = 2000** statement initializes the **bonus** variable.

Conditional Processing

The value assigned to **Bonus** is determined by testing for various values of **MPG Highway**.



9

The ELSE IF and ELSE Statement

Use the ELSE IF and ELSE statement to reproduce the if(), else if(), and else() functions in R.

```

data sp4r.cars;
set sp4r.cars;
if mpg_highway<20 then bonus=0;
else if mpg_highway<30 then bonus=1000;
else bonus=2000;
run;
  
```

*IF expression THEN statement;
<ELSE IF expression THEN statement;>
<...>
<ELSE statement;>*

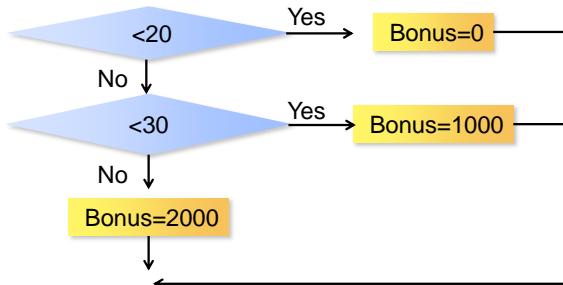
10

These statements are identical to the function names in R.

The **bonus = 1000** statement was removed in favor of the ELSE statement.

Conditional Processing

The ELSE IF and ELSE statements provide more efficient conditional processing.



11

Viewing the Output

```
proc print data=sp4r.cars (firstobs=76 obs=81);
  var mpg_highway bonus;
run;
```

PROC PRINT Output

Obs	MPG_Highway	bonus
76	32	2000
77	30	2000
78	28	1000
79	32	2000
80	28	1000
81	17	0

12

3.01 Multiple Choice Poll

Choose the correct statements. (Select all that apply.)

- a. The ELSE IF and ELSE statements provide more efficient conditional processing.
- b. The DATA step uses a SET statement to add new variables to the SAS data set.
- c. You do not need to initialize the new variable.

13

Conditional Processing for Character Variables

Suppose, in addition to the **Type** variable, a variable that indicates whether the vehicle is associated with being a family vehicle should also be in the data set. Create a new variable to account for the information below.

Type	Type2
Hybrid	Family Vehicle
SUV	Family Vehicle
Sedan	Family Vehicle
Wagon	Family Vehicle
Others	Truck or Sports Vehicle

15

Creating Character Variables

Use the LENGTH statement when you create character variables to avoid unwanted truncation.

```
data sp4r.cars;
  set sp4r.cars;
  length type2 $ 25;
  if type in ('Hybrid','SUV','Sedan','Wagon')
    then type2='Family Vehicle';
  else type2='Truck or Sports Vehicle';
run;
```

16

Although it is not necessary to use the LENGTH statement for variables that are less than eight characters, it is a good practice to use it anytime a character-valued variable is created.

Selected DATA step statement:

LENGTH specifies the length of each variable within the statement. The \$ symbol indicates that the new variable is a character value.

Viewing the Output

```
proc print data=sp4r.cars (firsttobs=61 obs=64);
  var type type2;
run;
```

PROC PRINT Output

Obs	Type	type2
61	Sedan	Family Vehicle
62	Sports	Truck or Sports Vehicle
63	Truck	Truck or Sports Vehicle
64	SUV	Family Vehicle

17

3.02 Poll

Suppose you are creating a new variable called **origin2** from the existing variable **origin**. Here you want to let **origin2** be *Asia* if **origin** is 'Asia'. Otherwise, let **origin2** be 'Foreign Country'. Does the DATA step below accomplish this task?

- True
- False

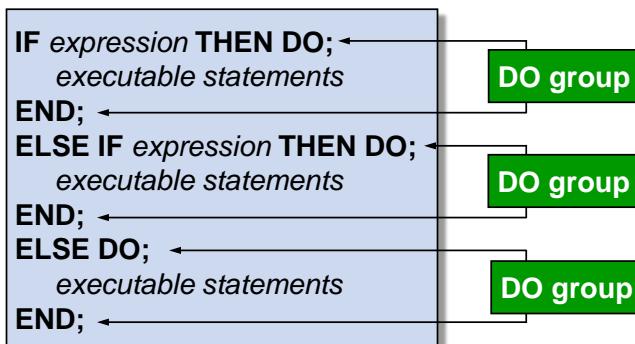
```
data sp4r.cars;
  set sp4r.cars;
  length origin2 $ 25;
  if origin='Asia' then origin2='Asia';
  else origin2='Foreign Country';
run;
```

18

Conditional DO Groups

DO groups enable SAS to execute ***multiple statements*** conditionally.

General form of conditional DO group syntax:



20

Conditionally Processing Multiple Statements

Suppose car manufacturers receive an economic incentive for manufacturing cars with high highway miles per gallon.

Create a new variable to reflect the bonus received and the frequency outlined in the table below.

MPG Highway	Bonus	Frequency
>=30	1000	Two Payments
20-29	1000	One Payment
<20	0	No Payment

21

Using the DO Group

```

data sp4r.cars;
  set sp4r.cars;
  length frequency $ 12;
  if mpg_highway<20 then do;
    bonus=0;
    frequency='No Payment';
  end;
  else if mpg_highway<30 then do;
    bonus=1000;
    frequency='One Payment';
  end;
  else do;
    bonus=1000;
    frequency='Two Payments';
  end;
run;

```

22

Viewing the Output

```
proc print data=cars (firstobs=65 obs=68);  
  var mpg_highway frequency;  
run;
```

PROC PRINT Output

Obs	MPG_Highway	bonus	frequency
65	18	0	No Payment
66	21	1000	One Payment
67	22	1000	One Payment
68	34	1000	Two Payments

23

3.03 Multiple Answer Poll

Choose the correct statements. (Select all that apply.)

- a. DO groups enable the execution of multiple statements.
- b. Each DO group ends with an END statement.
- c. It is a best practice to use a LENGTH statement when you create character variables.

24

3.2 Creating and Using Functions

Objectives

- Use SAS functions.
- Create and apply user-defined functions with the FCMP procedure.

27

SAS Functions

Use built-in SAS functions to assist in creating new variables.



28

Use SAS functions inside a DATA step to assist in new variable creation.

Duplicating the R Script

Create and use functions.

```
Source | 🔎 🖌️ | ⏷ ⏸ ⏹ ⏺ ⏻ ⏻ ⏻
#Create new function
mydivision = function(num,den){
  val = rep(NA,length(den))
  for(i in 1:length(den)){
    if(den[i]==0){val[i]=0}
    else {val[i]=round( num[i]/den[i] )}
  }
  return(val)
}
```

29

Idea Exchange

What built-in R function do you use most frequently?
Do you build user-defined functions to alter your data repeatedly?



30

SAS Functions

Use SAS functions within a DATA step to create new variables.

SAS Functions

SUM, MEAN, VAR, MEDIAN, MIN, MAX
EXP, LOG, SQRT, SIN, COS, TAN
ROUND, CEIL, FLOOR, ABS
+, -, *, /, **

```
data sp4r.cars;
  set sp4r.cars;
  log_price = log(msrp);
run;
```

new-variable = FUNCTION(arguments);

31

continued...

This list is by no means exhaustive. Search on **SAS functions** for a complete list.

- ✍ SAS does not use the \wedge symbol for exponentiation.

SAS Functions

SAS functions operate only on the rows of the data table.

SAS Functions

SUM, MEAN, VAR, MEDIAN, MIN, MAX
EXP, LOG, SQRT, SIN, COS, TAN
ROUND, CEIL, FLOOR, ABS
+, -, *, /, **

```
data sp4r.cars;
  set sp4r.cars;
  mean_mpg = mean(mpg_highway,mpg_city);
run;
```

32

continued...

The functions highlighted here behave differently than R functions. The R functions use the function on the entire vector of components whereas the SAS functions operate across rows in the data table.

Applying functions to columns is done using a SAS procedure, which is discussed in Chapter 5.

SAS Functions

Use a DATA _NULL_ step to avoid creating or altering a SAS data set.

```
data _NULL_;
  a=mean(1,2,3,4,5);
  b=exp(3);
  c=var(10,20,30);
  d=poisson(1,2);
  put a b c d;
run;
```

PUT variables;

SAS Log

```
3 20.085536923 100 0.9196986029
```

33

When you use the DATA _NULL_ statement, SAS processes the DATA step without writing observations to a data set. Using the DATA _NULL_ statement can increase program efficiency considerably.

The PUT statement writes lines to the SAS log.

3.04 Multiple Choice Poll

Which task does the DATA step below accomplish?
(Choose the correct statement.)

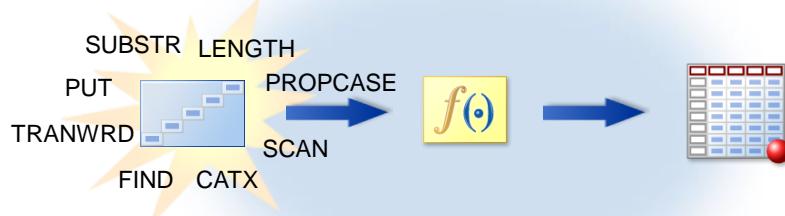
- Return the minimum value of **MPG_City**.
- Create a new variable that is an exact duplicate of **MPG_City**.

```
data sp4r.cars;
  set sp4r.cars;
  mpgvar=min(mpg_city);
run;
```

34

Manipulating Character Values

SAS has great functionality when you are working with character data.



36

Manipulating Character Values

SAS has great functionality when you are working with character data.

SAS Function	Description
SUBSTR	Extracts a substring from an argument.
PROPCASE, UPCASE, LOWCASE	Converts word casing.
SCAN	Returns the n^{th} word from a character string.
CATX	Removes leading and trailing blanks and concatenates character strings.
FIND	Searches for the location of a specific substring within a character string.
TRANWRD	Replaces all occurrences of a substring in a character string.
PUT	Returns a value using a specified format.

37

SUBSTR Function

Extract the fourth character from the value in the **Acct_Code** variable and store it in **Org_Code**.

```
NewVar = SUBSTR(string,start<,length);
```

```
Org_Code = substr(Acct_Code,4,1);
```

Acct_Code	Org_Code
AQI2	2

starting at position 4
for a length of 1

38

If *NewVar* is a new variable, it is created with the same length as the string. To set a different length for *NewVar*, use a LENGTH statement before the assignment statement in the DATA step.

- *String* can be a character constant, variable, or expression.
- *Start* specifies the starting position.
- *Length* specifies the number of characters to extract. If it is omitted, the substring consists of the remainder of *string*.

The SUBSTR function on the left side of an assignment statement is used to replace characters.

LENGTH Function

The LENGTH function returns the length of a non-blank character string, excluding trailing blanks.

```
NewVar = LENGTH(argument);
```

```
Org_Code=substr(Acct_Code,length(Acct_Code),1);
```

Acct_Code	Org_Code
AQI2	2
ES3	3
V2	2

39

3.05 Quiz

What would the variable location be after you use the SUBSTR function?

```
Location='Columbus, GA 43227';
substr(Location,11,2)='OH';
```

40

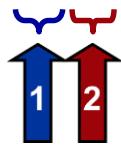
SCAN Function

Extract the second word of **Name**.

```
NewVar = SCAN(string,n<,charlist>);
```

```
FName = scan (Name,2,' ','');
```

Name	FName
Farr,Sue	Sue



The SCAN function returns the n^{th} word of a character value.

42

The SCAN function is used to extract words from a character value when the relative order of words is known, but their starting positions are not. The default delimiter is a blank.

When using the SCAN function, the following conditions exist:

- A missing value is returned if there are fewer than n words in the string.
- If n is negative, the SCAN function selects the word in the character string starting from the end of the string.
- The length of the created variable is the length of the first argument starting in SAS 9.4.
- The length of the created variable is 200 bytes in SAS 9.3 or earlier.

- Delimiters before the first words have no effect.
- Any character or set of characters can serve as a delimiter.
- Two or more contiguous delimiters are treated as a single delimiter.

CATX Function

Combine **FName** and **LName** to create **FullName**.

The CATX function removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string.

```
NewVar = CATX(separator, string-1, ... ,string-n)
```

```
FullName = catx(' ', FName, LName);
```

PDV

FName	LName	FullName
Sue	Farr	Sue Farr

43

TRANWRD Function

Change the error **Luci** to **Lucky**.

```
NewVar = TRANWRD(source,target,replacement);
```

```
Product = Tranwrd(Product,'Luci ','Lucky ');
```

Product_ID	Product	Order_ID
21 02 002 00003	Sunfit Trunks, Blue	1231986335
21 02 005 00003	Luci Knit Mittens, Red	1232003930
21 02 005 00004	Luci Knit mittens, blue	1232007693

Product_ID	Product	Order_ID
21 02 002 00003	Sunfit Trunks, Blue	1231986335
21 02 005 00003	Lucky Knit Mittens, Red	1232003930
21 02 005 00004	Lucky Knit mittens, blue	1232007693

44

The TRANWRD function replaces or removes all occurrences of a given word (or pattern of characters) within a character string.

- The TRANWRD function does not remove trailing blanks from *target* or *replacement*.
- If *NewVar* is not previously defined, it is given a length of 200.
- If the target string is not found in the source, then no replacement occurs.

3.06 Quiz

What is the value of the second observation of the **newname** variable in the **cars** data set if you run the DATA step below?

```
data sp4r.cars;
  set sp4r.cars;
  newname =
    upcase(catx(' ',make,scan(model,1)));
run;
```



Navigating to the SAS Online Functions Documentation

1. Go the SAS online documentation page at <http://support.sas.com/documentation/>.
2. Select **SAS 9.4** in the Knowledge Base pane.

KNOWLEDGE BASE

- [Products & Solutions](#)
- [System Requirements](#)
- [Install Center](#)
- [Third-Party Software Reference](#)
- **Documentation**
 - [What's New in SAS](#)
 - [Product Index A-Z](#)
 - **SAS 9.4**
 - [SAS Analytical Products 14.1](#)
 - [SAS Analytical Products 13.2](#)
 - [SAS Analytical Products 13.1](#)
 - [SAS 9.3](#)
 - [SAS Analytical Products 12.1](#)
 - [SAS 9.2](#)
 - [Earlier SAS Releases](#)
- [Papers](#)
- [Samples & SAS Notes](#)
- [Focus Areas](#)

3. Select **Programmer's Bookshelf** in the SAS 9.4 Product Documentation pane.

SAS 9.4 Product Documentation

Starting Points

- [Product Index A-Z](#)
- **Programmer's Bookshelf**
- [What's New in SAS](#)
- [Documentation by Title](#)

Syntax Shortcuts

- [Syntax Lookup](#)
- [SAS Procedures by Name and Product](#)
- [SAS Language Elements by Name, Product, and Category](#)

4. Select **Functions and CALL Routines** to expand the section.

Base SAS

- ⊕ Base SAS Procedures
- ⊕ Base SAS High-Performance Procedures
- ⊕ Base SAS Statistical Procedures
- ⊕ Language Reference: Concepts
 - The Little SAS Book: A Primer (book excerpt)
- ⊕ Component Objects
- ⊕ Data Set Options
- ⊕ Formats and Informats
- ⊕ Functions and CALL Routines
- ⊕ Functions by Example (book excerpt)
- ⊕ Statements

5. Select **SAS Functions and CALL Routines** to expand the section.

- What's New in SAS 9.4 Functions and CALL Routines
- About This Book
- ⊕ SAS Functions and CALL Routines
- ⊕ Dictionary of SAS Functions and CALL Routines
- ⊕ References
- Tables of Perl Regular Expression (PRX) Metacharacters
- Recommended Reading
- Functions by Example (book excerpt)

6. On the Contents tab, select **Dictionary of SAS Functions and CALL Routines** to expand the section and view a list of all SAS functions. They are sorted alphabetically.

Contents	About
<ul style="list-style-type: none"> What's New in SAS 9.4 Functions and CALL Routines About This Book ⊕ SAS Functions and CALL Routines ⊕ Dictionary of SAS Functions and CALL Routines ⊕ References Tables of Perl Regular Expression (PRX) Metacharacters Recommended Reading 	

7. Select **FIND Function** to view the syntax.

FIND Function

Searches for a specific substring of characters within a character string.

Category:	Character
Restriction:	This function is assigned an I18N Level 0 status, and is designed for SBCS data. Do not use this function to process DBCS or MBCS data. For more information, see Internationalization Compatibility .
Tip:	Use the KINDEX Function in SAS National Language Support (NLS) Reference Guide instead to write encoding independent code.

Syntax

- Required Arguments
- Optional Arguments

Details

- [Comparisons](#)
- [Example](#)
- [See Also](#)

8. Scroll to the bottom to view syntax examples.

Example

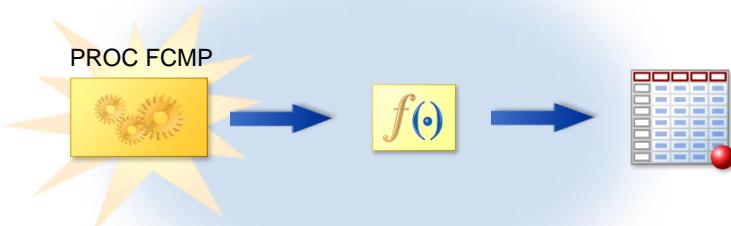
The following SAS statements produce these results:

SAS Statement	Result
whereisshe=find('She sells seashells? Yes, she does.','she '); put whereisshe;	27
variable1='She sells seashells? Yes, she does.'; variable2='she '; variable3='i'; whereisshe_i=find(variable1,variable2,variable3); put whereisshe_i;	1
expression1='She sells seashells? ' 'Yes, she does.'; expression2=kscan('he or she',3) ' '; expression3=trim('t '); whereisshe_t=find(expression1,expression2,expression3); put whereisshe_t;	14
xyz='She sells seashells? Yes, she does.'; startposvar=22; whereisshe_22=find(xyz,'she',startposvar); put whereisshe_22;	27
xyz='She sells seashells? Yes, she does.'; startposexp=1-23; whereisShe_ineg22=find(xyz,'She','i',startposexp); put whereisShe_ineg22;	14

End of Demonstration

PROC FCMP

Duplicate function creation in R with the function compiler procedure.



Create and maintain complex code to be used repeatedly.

48

PROC FCMP

Compare the function syntax between R and SAS.

```

R code:
function.name = function(arg1,arg2,...){
  programming.statements
  return(arguments)
}

SAS code:
PROC FCMP OUTLIB=libref.data-set.package;
  FUNCTION function-name(argument-1 <$>,...,
                        argument-n <$>) <$>; <length ,>
    programming-statements;
  RETURN(expression);
ENDSUB;

QUIT;
  
```

49

A function definition begins with the FUNCTION statement and ends with an ENDSUB statement.

A SAS function is a routine that accepts arguments, performs a computation or other operation, and returns either a character or numeric value. The syntax is highly similar to the R function. The FUNCTION statement is followed by the function name and the arguments in parentheses. In addition, each function uses the RETURN statement to identify the function output.

- A SAS function can return only a single value.

PROC FCMP Statement

The OUTLIB= option in the PROC FCMP statement specifies the three-level name of an output package to which the compiled functions are written.

```
proc fcmp outlib=work.functions.newfuncs;
  ...
quit; PROC FCMP OUTLIB=libref.data-set.package;
```

The **newfuncs** package is a collection of routines that have unique names and are stored in the **work.functions** data set.

50

A *package* is a collection of routines that have unique names. You can call the second and third argument of the OUTLIB option any name that you choose.

PROC FCMP

Create a new function to reverse the order of a name from last, first name to first, last name.

```
proc fcmp outlib=work.functions.newfuncs;
  function ReverseName(name $) $;
    length newname $ 40;
    newname=catx(' ',scan(name,2,''),scan(name,1,''));
    return(newname);
  endsub;
quit;
```

Character arguments must be followed by a dollar sign. In this example, the value returned by the ReverseName function is a character value with a maximum length of 40 characters.

51

Because the function returns a character value, the FUNCTION statement specifies this by the \$ symbol followed by the length.

Selected SAS functions:

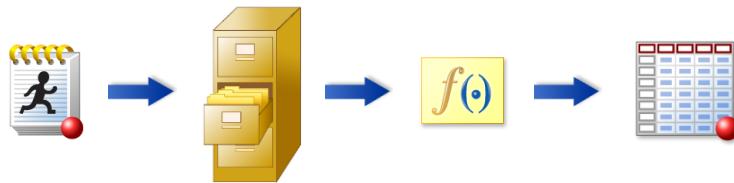
CATX removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string.

SCAN returns the n^{th} word from a character string.

Accessing the Newly Defined Function

The CMPLIB= SAS system option specifies one or more data sets that SAS searches for user-defined function entries. The default is **work.functions**.

```
options cmplib=work.functions;
CMPLIB=libref.data-set | (libref.data-set-1 ... libref.data-set-n)
```



52

Think of the CMPLIB option as the library statement in R. This statement effectively unpacks the functions in order to enable SAS to use the functions.

Accessing a User-Defined Function

Use the function in the DATA step to create **FLName**, which stores the rearranged name.

```
options cmplib=work.functions;
data sp4r.school;
  set sp4r.school;
  FLName=ReverseName(name);
run;
```

PDV

Name	...	FLName
Bakerman, Jordan	...	Jordan Bakerman

53

The OPTIONS statement specifies or changes the value of one or more SAS system options. For example, to suppress the data that is normally written to SAS output and set a line size of 72, execute the following statement:

```
options nodate linesize=72
```

Options are not saved. They must be run in each session.

3.07 Multiple Choice Poll

Choose the correct statements. (Select all that apply.)

- a. All built-in SAS functions operate the same as built-in R functions.
- b. PROC FCMP is the counterpart to the R function `function()`.
- c. The CMPLIB= option in the OPTIONS statement unpacks the user-defined function.

54

3.08 Quiz

You want to use PROC FCMP to create a function that avoids division by zero. If the divisor is zero, simply return a value of zero. What is missing from the PROC step below?

```
proc fcmp outlib=sp4r.functions.newfuncs;
  function perHomeRuns(num,den);
    if den = 0 then val = 0;
    else val = round(num/den);
  endsub;
quit;
```

56

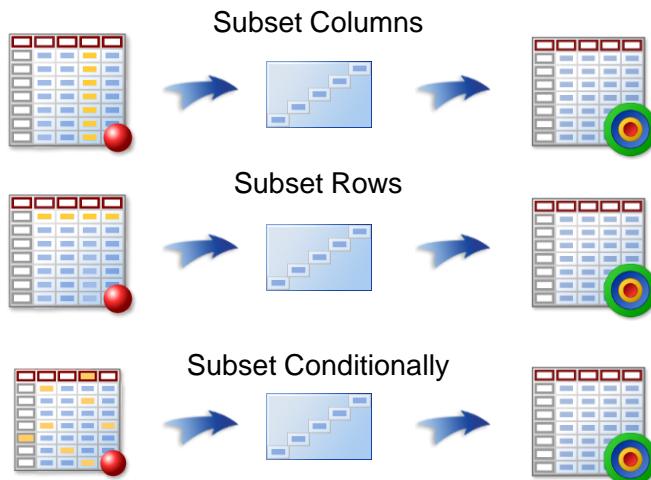
3.3 Subsetting and Concatenating Data Tables

Objectives

- Use the KEEP, DROP, FIRSTOBS, and OBS options in a DATA step to create a subsetted data table.
- Use the WHERE statement in a DATA step to create new data tables conditionally.
- Query the data table with PROC SQL to create new data tables.
- Use the SET statement in a DATA step to concatenate data tables .
- Use the MERGE and BY statements in a DATA step to merge data tables.

59

Motivation



60

Duplicating the R Script

Subset and concatenate data.

```
#Identify the unique Levels of the character valued variables
Make = levels(cars$Make)
Type = levels(cars>Type)
origin = levels(cars$origin)

#Create a list of the character valued variables
my_list = list(Make,Type,origin)

#Create a data frame for cars from Asia
Asia = cars[cars$origin=="Asia",]
Asia = data.frame(Asia>Name,Asia$Team,Asia$Home_Runs)

#Create a data frame for cars from Europe
Europe = cars[cars$origin=="Europe",]
Europe = data.frame(Europe>Name,Europe$Team,Europe$Home_Runs)

#Concatenate the Boston and Montreal data frames
bos_mon = rbind(Asia,Europe)
```

61

Subsetting by Column

Use the KEEP= option to select specific columns for a subsetted data table.

```
cars2 = data.frame(cars$Make,cars$MSRP, cars$Invoice)
```

```
data sp4r.cars2 (keep=make msrp invoice);
      set sp4r.cars;
run;
```

```
DATA new-data-table-name (KEEP=variable1 variable2 ...);
      SET old-data-table-name;
RUN;
```

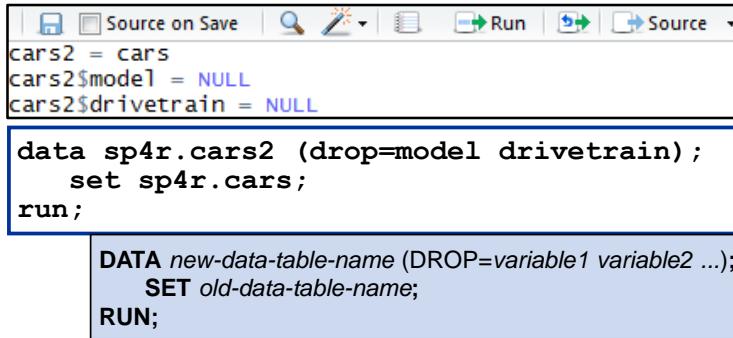
62

Selected DATA step options:

KEEP= specifies which variables to subset into the new data table from the data table that is specified in the SET statement.

Subsetting by Column

Use the DROP= option to remove specific columns for a subsetted data table.



```

cars2 = cars
cars2$model = NULL
cars2$drivetrain = NULL

data sp4r.cars2 (drop=model drivetrain);
  set sp4r.cars;
run;

DATA new-data-table-name (DROP=variable1 variable2 ...);
  SET old-data-table-name;
RUN;

```

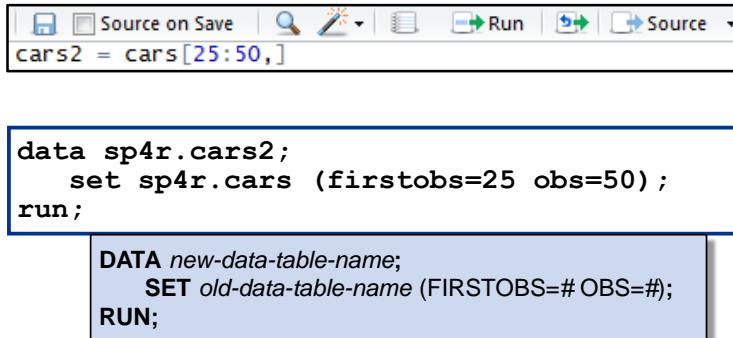
63

Selected DATA step options:

DROP= specifies which variables to exclude from the new data table from the data table that is specified in the SET statement.

Subsetting by Row

Use the FIRSTOBS= and the OBS= options to subset a group of observations.



```

cars2 = cars[25:50,]

data sp4r.cars2;
  set sp4r.cars (firstobs=25 obs=50);
run;

DATA new-data-table-name;
  SET old-data-table-name (FIRSTOBS=# OBS=#);
RUN;

```

64

Supplying the SET statement with the FIRSTOBS= and OBS= options specifies which range of observations to subset.

Subsetting Conditionally

Use the WHERE statement to subset a data table that is conditional on the observations.

```
Source on Save | Run | Source
cars2 = cars[cars$mpg_city > 35,]
```

```
data sp4r.cars2;
  set sp4r.cars;
  where mpg_city > 35;
run;
```

```
DATA new-data-table-name;
  SET old-data-table-name;
  WHERE conditional-expression;
RUN;
```

65

Selected DATA step statements:

WHERE specifies the conditions to subset the data.

The KEEP=, DROP=, FIRSTOBS=, and OBS= options can be combined with the WHERE statement to subset the data conditionally as well as according to column and row.

Subsetting by Query

Use PROC SQL to create a data table of the unique levels of a variable.

```
Source on Save | Run | Source
origin = Levels(cars$origin)
```

```
proc sql;
  create table sp4r.origin as
    select unique origin from sp4r.cars;
quit;
```

```
PROC SQL;
  CREATE TABLE new-data-table-name AS
    SELECT UNIQUE variable-name FROM old-data-table-name;
QUIT;
```

66

Select PROC SQL statements:

- CREATE TABLE requests SAS create a new data table with the proceeding name.
- AS refers SAS to the proceeding statement to be used for the query.
- SELECT UNIQUE requests a list of the unique levels from the proceeding variable name.
- FROM refers SAS to the desired data table to be queried.

- Multiple CREATE TABLE statements can be specified to create multiple data tables in a single SQL procedure.
- SELECT DISTINCT is identical to SELECT UNIQUE.

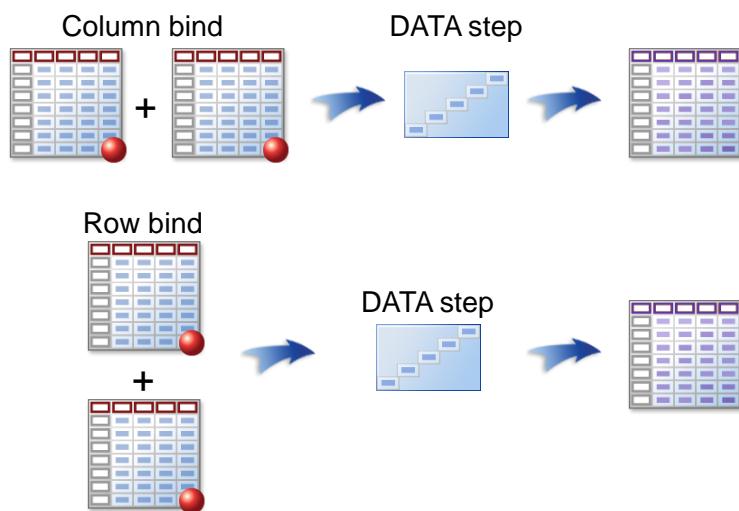
3.09 Multiple Choice Poll

Which statement and options are used to select column variables, rows, and conditional observations?

- a. SET, DROP, KEEP
- b. WHERE, FIRSTOBS= OBS=, SET
- c. KEEP, FIRSTOBS= OBS=, WHERE
- d. KEEP, WHERE, WHERE

67

Motivation

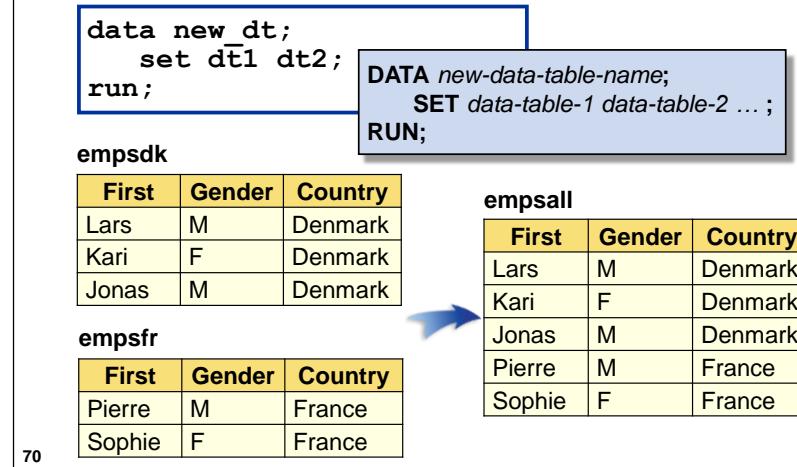


69

This section provides the tools to concatenate data tables in a manner similar to the rbind() and cbind() functions in R.

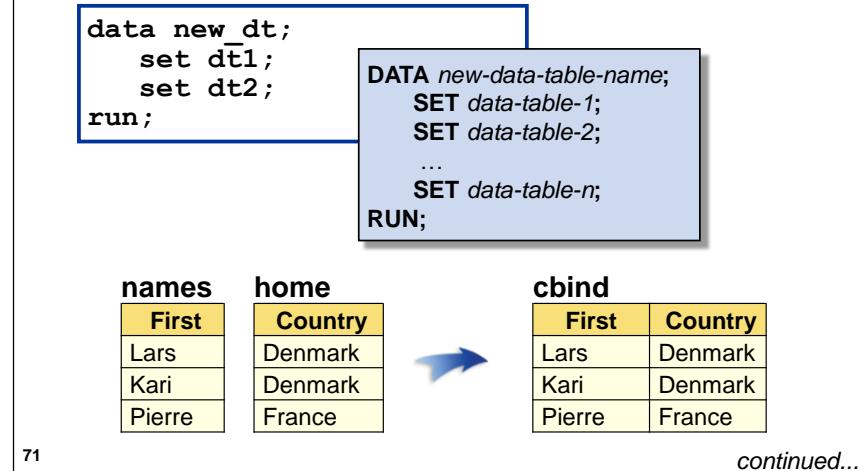
Row Bind Data Tables

To reproduce the rbind() function in R, supply the SET statement in a DATA step with multiple data tables.



Column Bind Data Tables

To reproduce the cbind() function in R, use multiple SET statements within a DATA step.

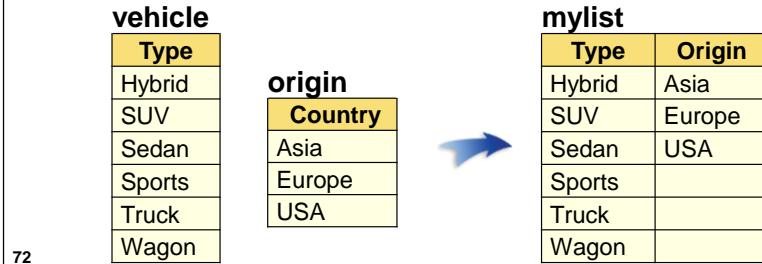


Using a SET statement to concatenate data tables of different dimensions removes observations without warning. The data table length is fixed at the length of the first data table provided in the SET statement.

Column Bind Data Tables

To concatenate data tables of different dimensions, use the MERGE statement within a DATA step.

```
data new_dt;
  merge dt1 dt2;
run;
DATA new-data-table-name;
  MERGE data-table-1 data-table-2 ...data-table-n;
RUN;
```



72

3.10 Multiple Choice Poll

Your colleague gave you three SAS data sets and wants you to combine them into one. The data sets are unique. This means that they are of different dimensions and contain different variables. Which DATA step statements should be used to combine these data sets?

- set dt1 dt2 dt3;
- merge dt1 dt2 dt3;
- set dt1; set dt2; set dt3;

73

Match-Merging

A	B	C		C	D	E
		1	←→	1		
		2	←→	2		
		3	←→	3		

One-to-One

A single observation in one data set is related to exactly one observation in another data set based on the values of one or more selected variables.

A	B	C		C	D	E
		1	←→	1		
		2	←→	1		
				2		

One-to-Many

A single observation in one data set is related to more than one observation in another data set based on the values of one or more selected variables.

A	B	C		C	D	E
		1	→	2		
		2	→	1		
				3		

Nonmatches

At least one observation in one data set is unrelated to any observation in another data set based on the values of one or more selected variables.

75

SORT, MERGE, and BY

Requirements for match-merging:

- Use PROC SORT to sort the data tables by the common variable that is being matched.
- List two or more tables in the MERGE statement.
- In the BY statement, list the variable that is common among data tables.

76

One-to-One Merge

One observation in **emps** matches exactly one observation in **phone**.

emps			phone	
First	Gender	EmpID	EmplD	Phone
Togar	M	121150	121150	+61(2)5555-1793
Kylie	F	121151	121151	+61(2)5555-1849
Birin	M	121152	121152	+61(2)5555-1665

Result:

emps			
First	Gender	EmpID	Phone
Togar	M	121150	+61(2)5555-1793
Kylie	F	121151	+61(2)5555-1849
Birin	M	121152	+61(2)5555-1665

77

One-to-Many Merge

One observation in **emps** matches one or more observations in **phone**.

emps			phones		
First	Gender	EmpID	EmplD	Type	Phone
Togar	M	121150	121150	Home	+61(2)5555-1793
			121150	Work	+61(2)5555-1794
Kylie	F	121151	121151	Home	+61(2)5555-1849
			121152	Work	+61(2)5555-1850
Birin	M	121152	121152	Home	+61(2)5555-1665
			121152	Cell	+61(2)5555-1666

78

One-to-Many Merge Result

empphones

First	Gender	EmpID	Type	Phone
Togar	M	121150	Home	+61(2)5555-1793
Togar	M	121150	Work	+61(2)5555-1794
Kylie	F	121151	Home	+61(2)5555-1849
Birin	M	121152	Work	+61(2)5555-1850
Birin	M	121152	Home	+61(2)5555-1665
Birin	M	121152	Cell	+61(2)5555-1666

79

Nonmatches Merge

There are observations in **emps** that do not have a match in **phone** and some in **phone** that do not match any observations in **emps**.

emps

First	Gender	EmpID
Togar	M	121150
Kylie	F	121151
Birin	M	121152

phone

EmpID	Phone
121150	+61(2)5555-1795
121152	+61(2)5555-1667
121153	+61(2)5555-1348

Result:

empsphone

First	Gender	EmpID	Phone
Togar	M	121150	+61(2)5555-1795
Kylie	F	121151	
Birin	M	121152	+61(2)5555-1667
		121153	+61(2)5555-1348

80



Exercises

Use the **Cars** data set in the **SP4R** library to complete the exercises.

1. Creating a New Data Set Variable

- Create a new variable called **mpg_average** in the **Cars** data set. This new variable should simply be the average gas mileage between **mpg_city** and **mpg_highway**.
- Print the first five observations for the variables **mpg_city**, **mpg_highway**, and **mpg_average** to ensure that the new variable is created.

2. Creating a New Data Set Variable Conditionally

- Use the new variable that you created in Exercise 1. Create a new variable in the **Cars** data set called **mpg_quality**, which is a character variable. Set **mpg_quality** according to the following table:

Mpg_average	Mpg_quality
<20	Low
20-29	Medium
>30	High

- Print observations 65 through 70 for the variables **mpg_average** and **mpg_quality** to ensure that the variable is created.

3. Creating and Using a SAS Function

- Create a function called **tier** with a single numeric argument, which returns a character value. The function should return values according to the following table:

input	output
<20	Low
20-29	Medium
>30	High

- Use the function that you created to create a new variable in the **Cars** data set. Name the new variable **mpg_quality2** and name the argument of the function **tier** as **mpg_average**. As a result, **mpg_quality** and **mpg_quality2** are identical.
- Print observations 65 through 70 for the variables **mpg_average**, **mpg_quality**, and **mpg_quality2** to ensure that the variable is created.

4. Creating a List of Unique Values

- Use PROC SQL to create three new data tables. Let values of **make** be the unique levels of the **make** variable. Let the values of **type** be the unique levels of the **type** variable. Let the values of **origin** be the unique levels of the **origin** variable.
- Create a new data table called **mylist**, which combines the three data tables.
Hint: This task requires you to column-bind data tables of different dimensions.

- c. Print **mylist** to ensure that the data table is created correctly.

Obs	Make	Type	Origin
1	Acura	Hybrid	Asia
2	Audi	SUV	Europe
3	BMW	Sedan	USA
4	Buick	Sports	
5	Cadillac	Truck	
6	Chevrolet	Wagon	
7	Chrysler		
8	Dodge		
9	Ford		
10	GMC		
11	Honda		
12	Hummer		
13	Hyundai		
14	Infiniti		
15	Isuzu		
16	Jaguar		
17	Jeep		
18	Kia		
19	Land Rover		
20	Lexus		
21	Lincoln		
22	MINI		
23	Mazda		
24	Mercedes-Benz		
25	Mercury		
26	Mitsubishi		
27	Nissan		
28	Oldsmobile		
29	Pontiac		
30	Porsche		
31	Saab		
32	Saturn		
33	Scion		
34	Subaru		
35	Suzuki		
36	Toyota		
37	Volkswagen		
38	Volvo		

5. Creating and Row-Binding Data Tables

- a. Create a new data table called **sports**, which has only three columns from the **Cars** data set: **make**, **type**, and **msrp**. In addition, keep only those observations where **type** is equal to *sports* and **msrp** is greater than \$100,000.
- b. Create another data table called **suv**, which has the same three columns. In addition, keep only those observations where **type** is equal to *suv* and **msrp** is greater than \$60,000.

- c. Create a new data table called **expensive** by row-binding **sports** and **suv**. Then print **expensive** to see the results.

Obs	Make	Type	MSRP
1	Mercedes-Benz	Sports	\$121,770
2	Mercedes-Benz	Sports	\$126,670
3	Porsche	Sports	\$192,465
4	Land Rover	SUV	\$72,250
5	Lexus	SUV	\$64,800
6	Mercedes-Benz	SUV	\$76,870

End of Exercises

3.4 Solutions

Solutions to Exercises

Use the **Cars** data set in the **SP4R** library to complete the exercises.

1. Creating a New Data Set Variable

- Create a new variable called **mpg_average** in the **Cars** data set. This new variable should simply be the average gas mileage between **mpg_city** and **mpg_highway**.

```
data sp4r.cars;
  set sp4r.cars;
  mpg_average = mean(mpg_city,mpg_highway);
run;
```

- Print the first five observations for the variables **mpg_city**, **mpg_highway**, and **mpg_average** to ensure that the new variable is created.

```
proc print data=sp4r.cars (obs=5);
  var mpg_city mpg_highway mpg_average;
run;
```

Obs	MPG_City	MPG_Highway	mpg_average
1	17	23	20.0
2	24	31	27.5
3	22	29	25.5
4	20	28	24.0
5	18	24	21.0

2. Creating a New Data Set Variable Conditionally

- Use the new variable that you created in Exercise 1. Create a new variable in the **Cars** data set called **mpg_quality**, which is a character variable. Set **mpg_quality** according to the following table:

Mpg_average	Mpg_quality
<20	Low
20-29	Medium
>30	High

```
data sp4r.cars;
  length mpg_quality $ 6;
  set sp4r.cars;
  if mpg_average < 20 then mpg_quality='Low';
  else if mpg_average < 30 then mpg_quality='Medium';
  else mpg_quality='High';
run;
```

- b. Print observations 65 through 70 for the variables **mpg_average** and **mpg_quality** to ensure that the variable is created.

```
proc print data=sp4r.cars (firstobs=65 obs=70);
  var mpg_average mpg_quality;
run;
```

Obs	mpg_average	mpg_quality
65	16.0	Low
66	18.5	Low
67	20.5	Medium
68	31.0	High
69	31.0	High
70	31.5	High

3. Creating and Using a SAS Function

- a. Create a function called **tier** with a single numeric argument, which returns a character value. The function should return values according to the following table:

input	output
<20	Low
20-29	Medium
>30	High

```
proc fcmp outlib=work.functions.newfuncs;
  function tier(val) $;
    length newval $ 6;
    if val < 20 then newval = 'Low';
    else if val <30 then newval='Medium';
    else newval='High';
    return(newval);
  endsub;
quit;
```

- b. Use the function that you created to create a new variable in the **Cars** data set. Name the new variable **mpg_quality2** and name the argument of the function **tier** as **mpg_average**. As a result, **mpg_quality** and **mpg_quality2** are identical.

```
options cmplib=work.functions;
data sp4r.cars;
  set sp4r.cars;
  mpg_quality2=tier(mpg_average);
run;
```

- c. Print observations 65 through 70 for the variables **mpg_average**, **mpg_quality**, and **mpg_quality2** to ensure that the variable is created.

```
proc print data=sp4r.cars (firstobs=65 obs=70);
  var mpg_average mpg_quality mpg_quality2;
run;
```

Obs	mpg_average	mpg_quality	mpg_quality2
65	16.0	Low	Low
66	18.5	Low	Low
67	20.5	Medium	Medium
68	31.0	High	High
69	31.0	High	High
70	31.5	High	High

4. Creating a List of Unique Values

- a. Use PROC SQL to create three new data tables. Let values of **make** be the unique levels of the **make** variable. Let the values of **type** be the unique levels of the **type** variable. Let the values of **origin** be the unique levels of the **origin** variable.

```
proc sql;
  create table make as select unique make from sp4r.cars;
  create table type as select unique type from sp4r.cars;
  create table origin as select unique origin from sp4r.cars;
quit;
```

- b. Create a new data table called **mylist**, which combines the three data tables.
Hint: This task requires you to column-bind data tables of different dimensions.

```
data sp4r.mylist;
  merge make type origin;
run;
```

- c. Print **mylist** to ensure that the data table is created correctly.

```
proc print data=sp4r.mylist;
run;
```

Obs	Make	Type	Origin
1	Acura	Hybrid	Asia
2	Audi	SUV	Europe
3	BMW	Sedan	USA
4	Buick	Sports	
5	Cadillac	Truck	
6	Chevrolet	Wagon	
7	Chrysler		
8	Dodge		
9	Ford		
10	GMC		
11	Honda		
12	Hummer		
13	Hyundai		
14	Infiniti		
15	Isuzu		
16	Jaguar		
17	Jeep		
18	Kia		
19	Land Rover		
20	Lexus		
21	Lincoln		
22	MINI		

```

23   Mazda
24   Mercedes-Benz
25   Mercury
26   Mitsubishi
27   Nissan
28   Oldsmobile
29   Pontiac
30   Porsche
31   Saab
32   Saturn
33   Scion
34   Subaru
35   Suzuki
36   Toyota
37   Volkswagen
38   Volvo

```

5. Creating and Row-Binding Data Tables

- a. Create a new data table called **sports**, which has only three columns from the **Cars** data set: **make**, **type**, and **msrp**. In addition, keep only those observations where **type** is equal to *sports* and **msrp** is greater than \$100,000.

```

data sp4r.sports(keep= make type msrp);
  set sp4r.cars;
  where type='Sports' and msrp>100000;
run;

```

- b. Create another data table called **suv**, which has the same three columns. In addition, keep only those observations where **type** is equal to *suv* and **msrp** is greater than \$60,000.

```

data sp4r.suv(keep= make type msrp);
  set sp4r.cars;
  where type='SUV' and msrp>60000;
run;

```

- c. Create a new data table called **expensive** by row-binding **sports** and **suv**. Then print **expensive** to see the results.

```

data sp4r.expensive;
  set sp4r.sports sp4r.suv;
run;

proc print data= sp4r.expensive;
run;

```

	Obs	Make	Type	MSRP
	1	Mercedes-Benz	Sports	\$121,770
	2	Mercedes-Benz	Sports	\$126,670
	3	Porsche	Sports	\$192,465
	4	Land Rover	SUV	\$72,250
	5	Lexus	SUV	\$64,800
	6	Mercedes-Benz	SUV	\$76,870

End of Solutions

Solutions to Student Activities (Polls/Quizzes)

3.01 Multiple Choice Poll – Correct Answer

Choose the correct statements. (Select all that apply.)

- a. The ELSE IF and ELSE statements provide more efficient conditional processing.
- b. The DATA step uses a SET statement to add new variables to the SAS data set.
- c. You do not need to initialize the new variable.

14

3.02 Poll – Correct Answer

Suppose you are creating a new variable called **origin2** from the existing variable **origin**. Here you want to let **origin2** be *Asia* if **origin** is 'Asia'. Otherwise, let **origin2** be 'Foreign Country'. Does the DATA step below accomplish this task?

- True
- False

```
data sp4r.cars;
  set sp4r.cars;
  length origin2 $ 25;
  if origin='Asia' then origin2='Asia';
  else origin2='Foreign Country';
run;
```

19

3.03 Multiple Answer Poll – Correct Answers

Choose the correct statements. (Select all that apply.)

- a. DO groups enable the execution of multiple statements.
- b. Each DO group ends with an END statement.
- c. It is a best practice to use a LENGTH statement when you create character variables.

25

3.04 Multiple Choice Poll – Correct Answer

Which task does the DATA step below accomplish?
(Choose the correct statement.)

- a. Return the minimum value of **MPG_City**.
- b. Create a new variable that is an exact duplicate of **MPG_City**.

```
data sp4r.cars;
  set sp4r.cars;
  mpgvar=min(mpg_city) ;
run;
```

35

3.05 Quiz – Correct Answer

What would the variable **location** be after you use the SUBSTR function?

```
Location='Columbus, GA 43227';
substr(Location,11,2)='OH';
```

Location
Columbus, OH 43227

41

3.06 Quiz – Correct Answer

What is the value of the second observation of the **newname** variable in the **cars** data set if you run the DATA step below?

```
data sp4r.cars;
  set sp4r.cars;
  newname =
    upcase(catx(' ',make,scan(model,1)));
run;
```

ACURA RSX

46

3.07 Multiple Choice Poll – Correct Answer

Choose the correct statements. (Select all that apply.)

- a. All built-in SAS functions operate the same as built-in R functions.
- b.** PROC FCMP is the counterpart to the R function function().
- c.** The CMPLIB= option in the OPTIONS statement unpacks the user-defined function.

55

3.08 Quiz – Correct Answer

You want to use PROC FCMP to create a function that avoids division by zero. If the divisor is zero, simply return a value of zero. What is missing from the PROC step below?

```
proc fcmp outlib=sp4r.functions.newfuncs;
  function perHomeRuns(num,den);
    if den = 0 then val = 0;
    else val = round(num/den);
    return(val);
  endsub;
quit;
```

57

3.09 Multiple Choice Poll – Correct Answer

Which statement and options are used to select column variables, rows, and conditional observations?

- a. SET, DROP, KEEP
- b. WHERE, FIRSTOBS= OBS=, SET
- c. KEEP, FIRSTOBS= OBS=, WHERE
- d. KEEP, WHERE, WHERE

68

3.10 Multiple Choice Poll – Correct Answer

Your colleague gave you three SAS data sets and wants you to combine them into one. The data sets are unique. This means that they are of different dimensions and contain different variables. Which DATA step statements should be used to combine these data sets?

- a. set dt1 dt2 dt3;
- b. merge dt1 dt2 dt3;
- c. set dt1; set dt2; set dt3;

74

Chapter 4 Random Number Generation and Plotting

4.1 DO Loop and Random Number Generation	4-3
Demonstration: DO Loop and Random Number Generation.....	4-13
4.2 Single-Cell Plotting.....	4-16
Demonstration: Creating Standard R Plots.....	4-25
Demonstration: Enhancing the Plot	4-35
4.3 Multi-Cell Plotting	4-38
Demonstration: Multi-Cell Plotting.....	4-45
Exercises	4-48
4.4 Solutions	4-55
Solutions to Exercises	4-55
Solutions to Student Activities (Polls/Quizzes)	4-66

4.1 DO Loop and Random Number Generation

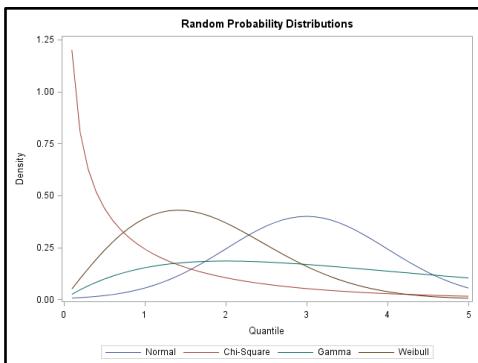
Objectives

- Use DO loop iterative processing.
- Use nested DO loops to create sequences and repetitive values.
- Create random probability distribution deviates with the RAND function.
- Use the PDF, CDF, and QUANTILE functions to identify the density, cumulative density, and quantile of a probability distribution value.

3

Motivation

Create data tables from random distributions.



Data Table



4

Duplicating the R Script

```
#Set a seed for random number generation
set.seed(123)

#Create a data set of random numbers
n=10
random = cbind(rnorm(n,20,5),rbinom(n,1,.25),runif(n,0,10),rexp(n,1/5))

#Add a random vector to the data set
random = cbind(random, rgeom(n,.1))

#Group random numbers
group = rep(1:5,each=3)
n=length(group)
random = cbind(group,1:n,rpois(n,25),rbeta(n,.5,.5))

#Find Density, CDF, and Quantile of distribution
q = seq(-3,3,by=.5)
d = dnorm(q,0,1)
p = pnorm(q,0,1)
random = cbind(q,d,p,qnorm(p,0,1))
```

5

DO Loop Processing

The DO loop is the foundation of the following:

- creating a sequence
- creating repetitive values
- creating groups
- random number generation
- creating new data



6

The DO loop is the key to creating a new data table. The number of loop iterations defines the table's row dimension and the number of variables defines the column dimension. The DO loop is used inside a DATA step to create new data tables or iterate through rows of an existing data table.

DO Loop Processing

The DO loop is equivalent to the seq() function in R.

```

seq(from=1,to=5)      do i=1 to 5;
seq(from=2,to=10,by=2) do i=2 to 10 by 2;
seq(from=10,to=2,by=-2)do i=10 to 2 by -2;

```

**DO index-variable=start TO stop <BY increment>;
END;**

7

DO Loop Processing

The number of DO loop iterations determines the number of observations that are written to the data table when you use the OUTPUT statement.

```

data loop;
  do i=2 to 10 by 2;
    x = i+1;
    rep = 1;
    output;
  end;
run;

```

**DATA data-table-name-new;
DO index-variable=start TO stop <BY increment>;
 iterated-SAS-statements;
OUTPUT;
END;
RUN;**

8

Every DO loop must include the DO and END statements. Optional statements are BY and OUTPUT. However, for data table creation, the OUTPUT statement should always be used.

Selected SAS statements:

- DO designates a group of statements to be executed as a unit. Initialize the loop with a variable name and an initial index value.
- TO specifies the final loop iteration value.
- BY increments the loop by the specified value. Omitting the BY statement causes the DO loop to iterate by 1.

OUTPUT requests all the iterated values be stored in the desired data table. One row is written to the table on each iteration of the DO loop.



Without an explicit OUTPUT statement, one implicit output occurs at the bottom of the DATA step and only one observation is written to the data table.

END terminates the DO loop.

DO Loop Processing

The OUTPUT statement writes all the values of all variables, including the index variable, to the data table.

```
data loop;
  do i=2 to 10 by 2;
    x = i+1;
    rep = 1;
    output;
  end;
run;
```

PROC PRINT Output

Obs	i	x	rep
1	2	3	1
2	4	5	1
3	6	7	1
4	8	9	1
5	10	11	1

9

DO Loop Processing

Use the KEEP= and DROP= options to control which variables are written to the new data table.

```
data loop (keep=x rep);
  do i=2 to 10 by 2;
    x = i+1;
    rep = 1;
    output;
  end;
run;
```

```
data loop (drop=i);
  do i=2 to 10 by 2;
    x = i+1;
    rep = 1;
    output;
  end;
run;
```

PROC PRINT Output

Obs	x	rep
1	3	1
2	5	1
3	7	1
4	9	1
5	11	1

10

Both DATA steps produce the same data table. Use the KEEP or DROP statement depending on the ease of variable specification.

Selected SAS statements:

KEEP requests that only the specified variables be written to the data table.

DROP requests that all variables, except the specified variables, be used to create the data table.

Nested DO Loops

Duplicate the rep() function in R with a nested DO loop.

```
Source on Save | Run | Source
rep(1:2,each=2)
rep(1:2,2)
```

```
do i=1 to 2;
  do j=1 to 2;
    output;
  end;
end;
```

PROC PRINT Output

Obs	i	j
1	1	1
2	1	2
3	2	1
4	2	2

11

A nested DO loop can be used to replicate the predecessor DO loop variables and to create groups. Here, the first variable *i* is replicated *j* times. The variable *i* acts as the grouping variable and the variable *j* acts as the observation number in the specified group.

Nested DO Loops

Applying a DO loop to an existing data table has the same effect as a nested DO loop.

- The DO loop iterates through each row of the data table.

```
data doloop;
  do i=1 to 2;
    output;
  end;
run;
```

```
data doloop;
  set doloop;
  do j=1 to 2;
    output;
  end;
run;
```

PROC PRINT Output

Obs	i
1	1
2	2

Obs	i	j
1	1	1
2	1	2
3	2	1
4	2	2

12

The DO loop can be applied to an existing data table to replicate the data.

Nested DO Loops

Use a SUM statement to add a sequence to

- an existing data table
- a nested DO loop.

```
data doloop;
  do i=1 to 2;
    do j=1 to 2;
      seq + 1;
      output;
    end;
  end;
run;
```

new-variable-name + sequence-value;

PROC PRINT Output →

Obs	i	j	seq
1	1	1	1
2	1	2	2
3	2	1	3
4	2	2	4

13

The SUM statement creates a new variable. The variable is automatically initialized to zero and its value is retained from one iteration of the DATA step to the next. On each iteration, the new variable is incremented by the sequence value. The SUM statement can be useful when you add a sequence to a data table.

4.01 Poll

Does the following DO loop create a data table with a sequence from 50 to 100 by 5 with the variable name **myloop**?

- Yes
- No

```
data doloop;
  do myloop=50 to 100 by 5;
  end;
run;
```

14

4.02 Multiple Choice Poll

What is the dimension of the data set created below?

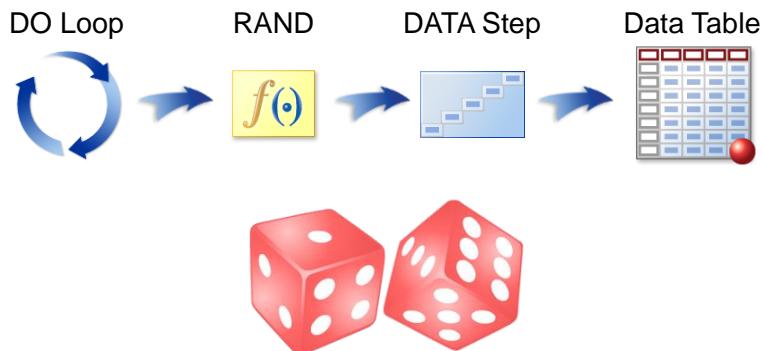
- a. 12x2
- b. 6x3
- c. 8x3
- d. 12x3

```
data random;
  do i=1 to 3;
    do j=1 to 2;
      do k=1 to 2;
        output;
      end;
    end;
  run;
```

16

Random Number Generation

Use the DO loop to sample from probability distributions.



18

The DO loop is used to simultaneously create a data table and generate random numbers. The DO loop specifies how many random numbers to generate.

Random Number Generation

Use the RAND function to sample from probability distributions.

```
RAND('distribution',param-1,param-2,...);
```

R	SAS
rbinom(n,size,p)	RAND('Binomial',p,n)
rexp(n,rate)	RAND('Exponential')
rnorm(n,mean,sd)	RAND('Normal',mean,sd)
rpois(n,mean)	RAND('Poisson',mean)
runif(n,min,max)	RAND('Uniform')

19

This is a sample list comparing the syntax between R and SAS. Search on **SAS RAND function** to view an exhaustive list. The RAND function's first input is the distribution name in quotation marks followed by the distribution parameters.

Random Number Generation

Generate three values from a Normal distribution with a mean of 10 and a standard deviation of 2.

- Use the CALL STREAMINIT option to set a seed.

```
data random (drop=i);
  call streaminit(123) ; CALL STREAMINIT(seed);
  do i=1 to 3;
    x = rand('Normal',10,2);
    output;
  end;
run;
```

20

Use the CALL STREAMINIT routine before the DO loop to set a desired seed.

The number of random samples is specified by the DO loop, not the RAND function. Here you are generating only three random deviates.

Random Number Generation

Add variables of random deviates to an existing data table.

- Use the RAND function without the DO loop.

```
data sp4r.cars;
  call streaminit(123) ; CALL STREAMINIT(seed);
  set sp4r.cars;
  x = rand('Normal',10,2) ;
run;                                RAND('distribution',param-1,param-2,...);
```

- The RAND function generates the same number of random deviates as observations in the existing data table.

21

Other Probability Functions

The functions that are used to return the density, cumulative density, and quantile of a probability distribution have similar syntax as the RAND function.

- PDF

```
PDF('distribution',quantile, param-1,param-2,...)
```

- CDF

```
CDF('distribution',quantile, param-1,param-2,...)
```

- QUANTILE

```
QUANTILE('dist',probability, param-1,param-2,...)
```

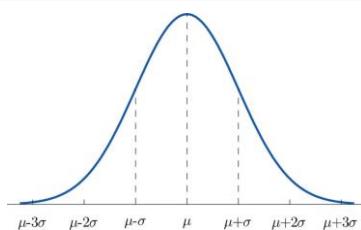
22

Other Probability Functions

Compare the PDF, CDF, and QUANTILE functions with the R counterparts.

- Use the Normal distribution as an example.

R	SAS
<code>dnorm(q,mean,sd)</code>	<code>PDF('Normal',q,mean,sd)</code>
<code>pnorm(q,mean,sd)</code>	<code>CDF('Normal',q,mean,sd)</code>
<code>qnorm(p,mean,sd)</code>	<code>QUANTILE('Normal',p,mean,sd)</code>



23



You can also use these functions in a DATA _NULL_ step to print the results of these SAS functions to the log.

4.03 Poll

Do the SAS functions `rand('Beta',5,7)` and `CDF('Beta',.3,5,7)` reproduce the R functions `rbeta(1,5,7)` and `pbeta(.3,5,7)`?

- Yes
- No

24



DO Loop and Random Number Generation

SP4R04d01.sas

- Create a data table with 10 random deviates from four different probability distributions. Use the random seed **123** and drop the DO loop index variable. Let the distributions be a normal distribution with a mean of 20 and standard deviation of 5, a Bernoulli distribution with a probability of 0.25, a uniform distribution from 0 to 10, and an exponential distribution with a mean of 5. Print the data table.



The uniform distribution and the exponential distribution have no inputs.

```
data sp4r.random (drop=i);
  call streaminit(123);
  do i=1 to 10;
    rnorm = rand('Normal',20,5);
    rbinom = rand('Binomial',.25,1);
    runif = rand('Uniform')*10;
    rexp = rand('Exponential')*5;
    output;
  end;
run;

proc print data=sp4r.random;
run;
```

Obs	rnorm	rbinom	runif	rexp
1	20.4197	0	3.87845	5.55589
2	17.6145	0	3.37595	8.88334
3	19.8906	1	8.46208	5.39953
4	22.1470	1	0.18005	4.14268
5	24.5320	1	7.51251	3.26894
6	25.0477	1	4.59901	0.40410
7	19.8704	0	2.95333	2.64741
8	21.0267	1	9.21826	2.31966
9	20.3133	1	5.06738	0.52389
10	22.7076	0	7.32552	1.20990

- Add an additional variable to the data table above. Generate random numbers from a geometric distribution with a probability parameter of 0.1. Use the same random seed of 123. Print the data table upon completion.

```
data sp4r.random;
  call streaminit(123);
  set sp4r.random;
  rgeom = rand('Geometric',.1);
run;

proc print data=sp4r.random;
run;
```

Obs	rnorm	rbinom	runif	rexp	rgeom
1	20.4197	0	3.87845	5.55589	2
2	17.6145	0	3.37595	8.88334	1
3	19.8906	1	8.46208	5.39953	6
4	22.1470	1	0.18005	4.14268	1
5	24.5320	1	7.51251	3.26894	9
6	25.0477	1	4.59901	0.40410	4
7	19.8704	0	2.95333	2.64741	33
8	21.0267	1	9.21826	2.31966	4
9	20.3133	1	5.06738	0.52389	1
10	22.7076	0	7.32552	1.20990	1

3. Create a data table with 15 random deviates from two different probability distributions. Use the random seed 123. Let the distributions be Poisson with a mean of 25 and a Beta with parameters 0.5 and 0.5. Group these 15 observations into five different groups of three observations each. Finally, create a sequence from 1 to 15 to be included in the data table. Print the data table upon completion.

```

data sp4r.doloop (drop=j);
  call streaminit(123);
  do group=1 to 5;
    do j=1 to 3;
      rpois = rand('Poisson',25);
      rbeta = rand('Beta',.5,.5);
      seq+1;
      output;
    end;
  end;
run;

proc print data=sp4r.doloop;
run;

```

Obs	group	rpois	rbeta	seq
1	1	25	0.95447	1
2	1	31	0.73901	2
3	1	30	0.07951	3
4	2	24	0.00319	4
5	2	22	0.27194	5
6	2	31	0.42317	6
7	3	29	0.94307	7
8	3	20	0.98216	8
9	3	26	0.30177	9
10	4	26	0.97667	10
11	4	15	0.08009	11
12	4	27	0.57148	12
13	5	29	0.03174	13
14	5	28	0.97330	14
15	5	19	0.00528	15

4. Use a DO loop to create quantiles from -3 to 3 by 0.5. This creates 13 iterations. For the remaining arguments, use a normal distribution with a mean of 0 and a standard deviation of 1. For each iteration, identify the density and the cumulative density and create new variables, **PDF** and **CDF**. Finally, use the new **CDF** variable to create a quantile variable that mirrors the DO loop values. Print the data table upon completion.

```

data sp4r.quants;
do q=-3 to 3 by .5;
  pdf = pdf('Normal',q,0,1);
  cdf = cdf('Normal',q,0,1);
  quantile = quantile('Normal',cdf,0,1);
  output;
end;
run;

proc print data=sp4r.quants;
run;

```



You can use function names as variable names in SAS.

Obs	q	pdf	cdf	quantile
1	-3.0	0.00443	0.00135	-3.0
2	-2.5	0.01753	0.00621	-2.5
3	-2.0	0.05399	0.02275	-2.0
4	-1.5	0.12952	0.06681	-1.5
5	-1.0	0.24197	0.15866	-1.0
6	-0.5	0.35207	0.30854	-0.5
7	0.0	0.39894	0.50000	0.0
8	0.5	0.35207	0.69146	0.5
9	1.0	0.24197	0.84134	1.0
10	1.5	0.12952	0.93319	1.5
11	2.0	0.05399	0.97725	2.0
12	2.5	0.01753	0.99379	2.5
13	3.0	0.00443	0.99865	3.0

End of Demonstration

4.2 Single-Cell Plotting

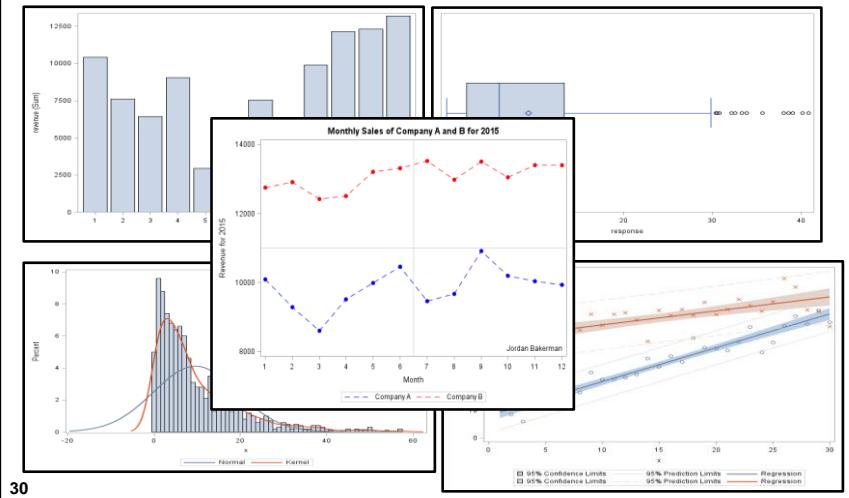
Objectives

- Use the SGPlot procedure to create single cell plots of all types.
- Enhance the visual presentation of the plot.

29

Motivation

Duplicate the base R plotting capabilities.



30

The four background plots denote easy-to-make plots without visual enhancement. You explore generating simple plots and progress to creating presentation-worthy plots.

Duplicating the R Script

31

```

#Create histogram with density estimate
n=300; vec = rexp(n,1/10); s = rep(1:3,each=100)
hist(vec, 50,freq=F)
lines(density(vec),col="red")

#Create boxplot
boxplot(vec~s,horizontal=T)

#Create bar chart
n=12; vec = rnorm(12,10000,5000)
barplot(vec)

#Create scatter plot
x = 1:30; y1 = 10+x+rnorm(30); y2 = 35+x/2+rnorm(30)
plot(y1~x,ylim=c(10,50)); abline(lm(y1~x))
points(y2~x); abline(lm(y2~x))

#Enhance the Plot
x = 1:12
revenue = rnorm(12,10000,1000)
revenue_2 = rnorm(12,13000,500)
plot(revenue~x,type="b",col="blue",ylim=c(8000,14000),
     main="Monthly sale of company A and B for 2015",
     xlab="Month",ylab="Revenue for 2015",pch=16,lty=2)
lines(revenue_2~x,type="b",col="red",pch=16,lty=2)
text(10,8000,"Jordan B")
abline(h=11000,col="grey"); abline(v=6.5,col="grey")
legend(2,9500,c("A","B"),col=c("blue","red"),lty=c(2,2))

```

Idea Exchange

What types of plots do you make in R?

32



The SGPlot Procedure

The base R plotting capabilities can be accomplished with the SGPLOT procedure to accomplish the following tasks:

- produce single-cell plots
- overlay plots on a single set of axes
- enhance the presentation of the plot

33

The SGPlot Procedure

The SGPLOT procedure statements can be organized into the following four categories:

TYPE	Plots	PLOT Statement
Basic	scatter, series, step, needle, vector, bubble, and band	SCATTER, SERIES, STEP, NEEDLE, VECTOR, BUBBLE, BAND
Fit and Confidence	regression, loess, penalized B-spline curves, ellipses	REG, LOESS, PBSPLINE, ELLIPSE
Distribution	box plot, histogram, normal/kernel density	HBOX, VBOX, HISTOGRAM, DENSITY
Categorical	bar chart, line chart, and dot plots	HBAR, VBAR, HLINE, VLINE, DOT

34

Selected PROC SGLOT statements:

SCATTER	creates a scatter plot of observations for two variables.
SERIES	creates a line plot; useful where X-axis values are related, such as time.
STEP	creates a step plot; useful where Y-axis values reflect a step function, such as integers.
NEEDLE	creates a plot with needles connecting each point to the baseline; useful for showing deviations from a constant.
VECTOR	creates a vector plot that draws arrows from a point of origin to each data point; useful for showing bivariate deviations from an origin such as a centroid.
BUBBLE	creates a bubble plot; useful where an additional variable reflects the size of the scatter plot points.
BAND	creates a band that highlights part of the plot; useful for specifying a range within the plot.
REG	creates a fitted regression line or curve to a scatter plot. Enables visualization of a least squares fit.
ELLIPSE	Adds a prediction or confidence ellipse to a plot. Enables visualization of a region where observations are likely to fall. Prediction is the default.
LOESS	creates a fitted loess curve; useful for visualizing a nonlinear relationship between two variables. As an option, smoothness can be controlled by a smoothing parameter and polynomial degree.
PBSPLINE	creates a fitted penalized B-spline curve; useful for visualizing a nonlinear relationship between two variables. As an option, smoothness can be controlled by a smoothing parameter and the number of knots.
HBOX	creates a horizontal box plot; useful for comparing distributions of grouped data.
VBOX	creates a vertical box plot; useful for comparing distributions of grouped data.
HISTOGRAM	creates a histogram that displays the frequency or percent of a numeric variable.
DENSITY	creates a density curve that shows the distribution of values for a numeric variable. Overlay a density curve on a histogram. Specify a normal distribution or a kernel estimate.
HBAR	creates a horizontal bar chart.
VBAR	creates a vertical bar chart.
HLINE	creates a horizontal line chart; useful when the y-axis values are ordered.
VLINE	creates a vertical line chart; useful when the x-axis values are ordered.
DOT	creates a dot plot.

The SGPLOT Procedure

The SGPLOT procedure statements conform to different syntax depending on the plot type.

TYPE	Plots	PLOT Statement
Basic	scatter, series, step, needle, vector, bubble, and band	SCATTER, SERIES, STEP, NEEDLE, VECTOR, BUBBLE, BAND
Fit and Confidence	regression, loess, penalized B-spline curves, ellipses	REG, LOESS, PBSPLINE, ELLIPSE
Distribution	box plot, histogram, normal/kernel density	HBOX, VBOX, HISTOGRAM, DENSITY
Categorical	bar chart, line chart, and dot plots	HBAR, VBAR, HLINE, VLINE, DOT

PLOTNAME X=x-variable Y=y-variable </ OPTIONS>;

35

For the Basic and Fit-and-Confidence plot types, specify the PLOT statement followed by the X-axis variable and the Y-axis variable.

The SGPROCEDURE Procedure

The SGPROCEDURE procedure statements conform to different syntax depending on the plot type.

TYPE	Plots	PLOT Statement
Basic	scatter, series, step, needle, vector, bubble, and band	SCATTER, SERIES, STEP, NEEDLE, VECTOR, BUBBLE, BAND
Fit and Confidence	regression, loess, penalized B-spline curves, ellipses	REG, LOESS, PBSPLINE, ELLIPSE
Distribution	box plot, histogram, normal/kernel density	HBOX, VBOX, HISTOGRAM, DENSITY
Categorical	bar chart, line chart, and dot plots	HBAR, VBAR, HLINE, VLINE, DOT

PLOTNAME response-or-category-variable </ OPTIONS>;

36

For the Distribution and Categorical plot types, specify the PLOT statement followed by the response variable.

The SGPLOT Procedure

The SGPLOT procedure statements conform to different syntax depending on the plot type.

TYPE	Plots	PLOT Statement
Basic	scatter, series, step, needle, vector, bubble, and band	SCATTER, SERIES, STEP, NEEDLE, VECTOR, BUBBLE, BAND
Fit and Confidence	regression, loess, penalized B-spline curves, ellipses	REG, LOESS, PBSPLINE, ELLIPSE
Distribution	box plot, histogram, normal/kernel density	HBOX, VBOX, HISTOGRAM, DENSITY
Categorical	bar chart, line chart, and dot plots	HBAR, VBAR, HLINE, VLINE, DOT

BAND X=x-variable LOWER=lower-bound UPPER=upper-bound;

37

For a Band plot, specify the X-axis variable followed by the lower and upper region to be filled.

The SGPLOT Procedure

The SGPLOT procedure statements conform to different syntax depending on the plot type.

TYPE	Plots	PLOT Statement
Basic	scatter, series, step, needle, vector, bubble , and band	SCATTER, SERIES, STEP, NEEDLE, VECTOR, BUBBLE, BAND
Fit and Confidence	regression, loess, penalized B-spline curves, ellipses	REG, LOESS, PBSPLINE, ELLIPSE
Distribution	box plot, histogram, normal/kernel density	HBOX, VBOX, HISTOGRAM, DENSITY
Categorical	bar chart, line chart, and dot plots	HBAR, VBAR, HLINE, VLINE, DOT

PLOTNAME X=x-variable Y=y-variable SIZE=size-variable;

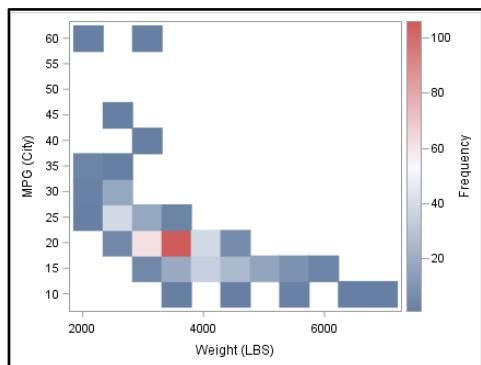
38

For a bubble plot, specify the x-axis variable, y-axis variable, and a numeric variable to alter the size of the scatter plot points.

4.04 Multiple Choice Poll

Navigate to the SGPlot procedure Help documentation and examine the plotting statements. Which statement is used to create the following plot?

- a. HEATMAP
- b. POLYGON
- c. BUBBLE
- d. BLOCK

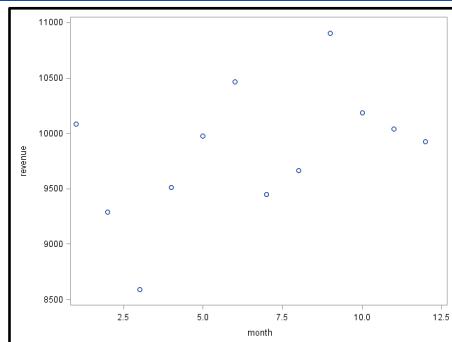


39

SGPLOT Procedure

Create a scatter plot of company revenue for the past 12 months.

```
proc sgplot data=sales;
  scatter x=month y=revenue;
run;
```



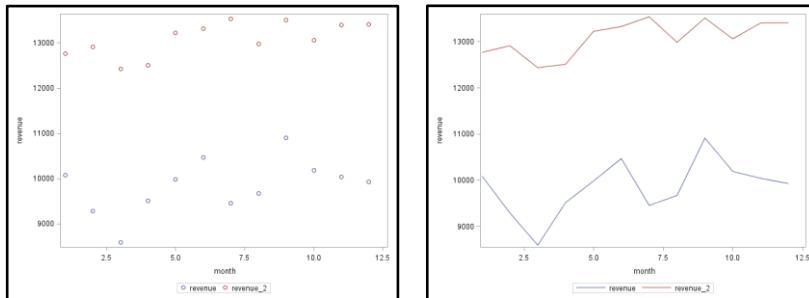
41

The minimum plotting syntax produces an identical plot to R. The plot produces axis labels according to the variable names.

SGPLOT Procedure

Reproduce the points() and lines() functions in R with multiple SCATTER or SERIES statements.

```
proc sgplot data=sales;
  scatter x=month y=revenue;
  scatter x=month y=revenue_2;
run;
```



42

The plot on the left uses the SCATTER statement and the plot on the right uses the SERIES statement. Overlaying occurs within the same procedure unlike the addition of multiple functions in R.

The SGPlot procedure automatically populates a legend when multiple SCATTER or SERIES statements are provided.

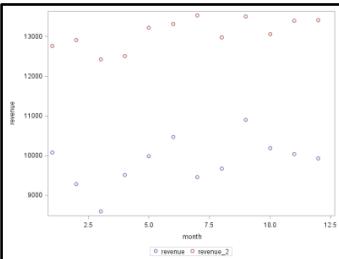
SGPLOT Procedure

An alternative overlay approach depends on the structure of the data table.

- If the response variables are stacked, use the GROUP= option.

```
proc sgplot data=sales;
  scatter x=month y=revenue / group=company;
run;
```

Obs	company	month	revenue
1	1	1	10083.94
2	1	2	9287.52
...			
12	1	12	9923.39
13	2	1	12761.45
14	2	2	12905.25
...			
24	2	12	13403.48



43

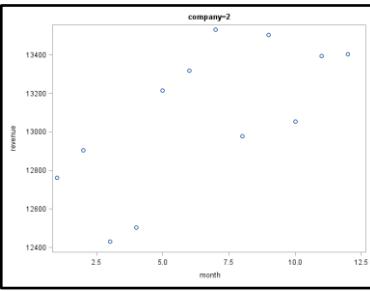
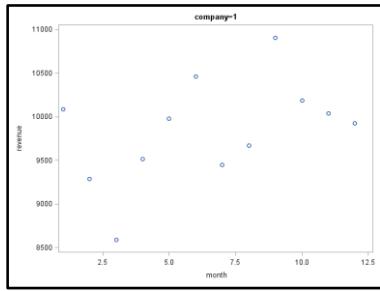
The GROUP= option overlays the same plot type for each category of the group variable. Here the Revenue variable is stacked and the Company variable refers to either company 1 or company 2. The previous slide assumed that the revenue was split into variables Revenue and Revenue_2.

SGPLOT Procedure

If the response variables are stacked, use the BY= statement to produce a plot for each classification.

```
proc sgplot data=sales;
  scatter x=month y=revenue;
  by company;
run;
```

BY classification-variable;



44

The BY statement produces a plot for each category of the specified variable. Use the GROUP= option to overlay and the BY statement to output multiple plots.



Creating Standard R Plots

SP4R04d02.sas

1. Create a data table of 1000 random deviates from an exponential distribution with a mean of 10. Create a histogram of the new variable.

```

data sp4r.hist_data;
  call streaminit(123);
  do i=1 to 1000;
    x = rand('exponential')*10;
    output;
  end;
run;

proc sgplot data=sp4r.hist_data;
  histogram x;
run;

proc sgplot data=sp4r.hist_data;
  histogram x / binwidth=1;
  density x / type=normal;
  density x / type=kernel;
run;

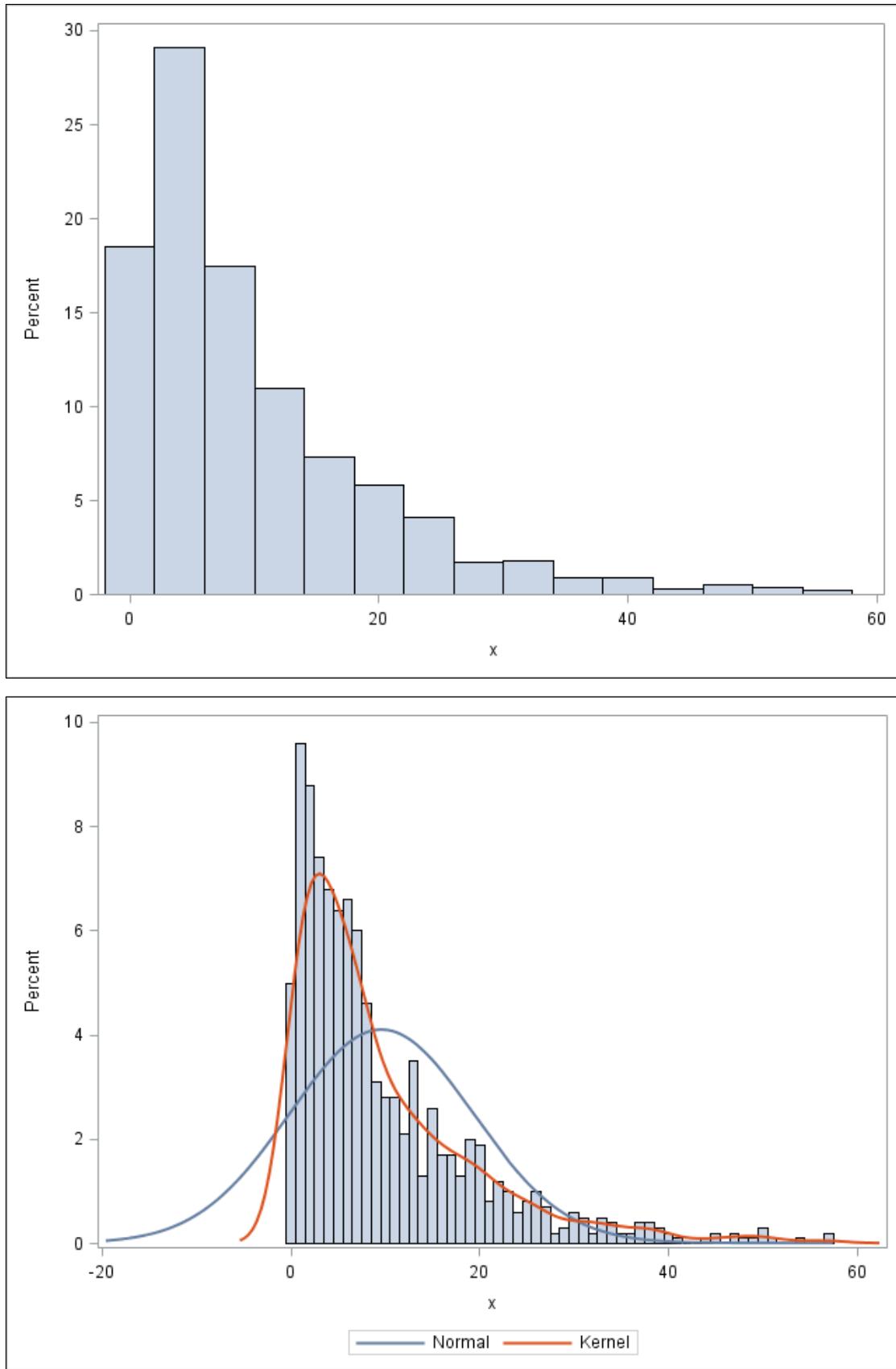
```

The density estimate is overlaid on the histogram when both statements appear in the SGPlot procedure.

Selected PROC SGPlot statements and options:

HISTOGRAM creates a histogram of the proceeding variable. Use the BINWIDTH= option to alter the bin size.

DENSITY creates a density estimate of the proceeding variable. Use the TYPE= option to estimate the normal distribution or the kernel distribution.



2. Create a new data table with three groups of 100 observations each. Let each observation be a random deviate from an exponential distribution with a mean of 10. First, create a horizontal box plot that ignores the **group** variable. Second, create a horizontal box plot with the **CATEGORY=** option.

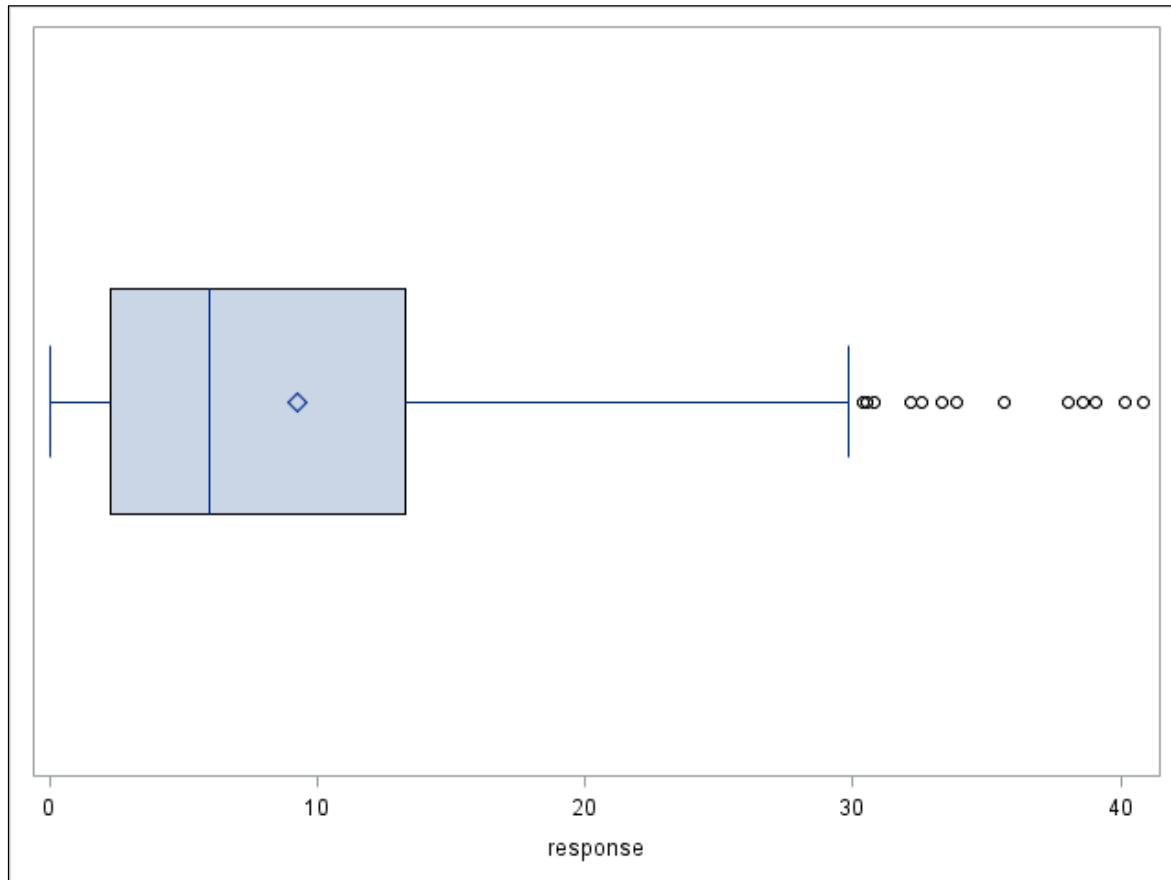
```
data sp4r.boxplot_data (drop=rep);
  call streaminit(123);
  do group=1 to 3;
    do rep=1 to 100;
      response = rand('exponential')*10;
      output;
    end;
  end;
run;

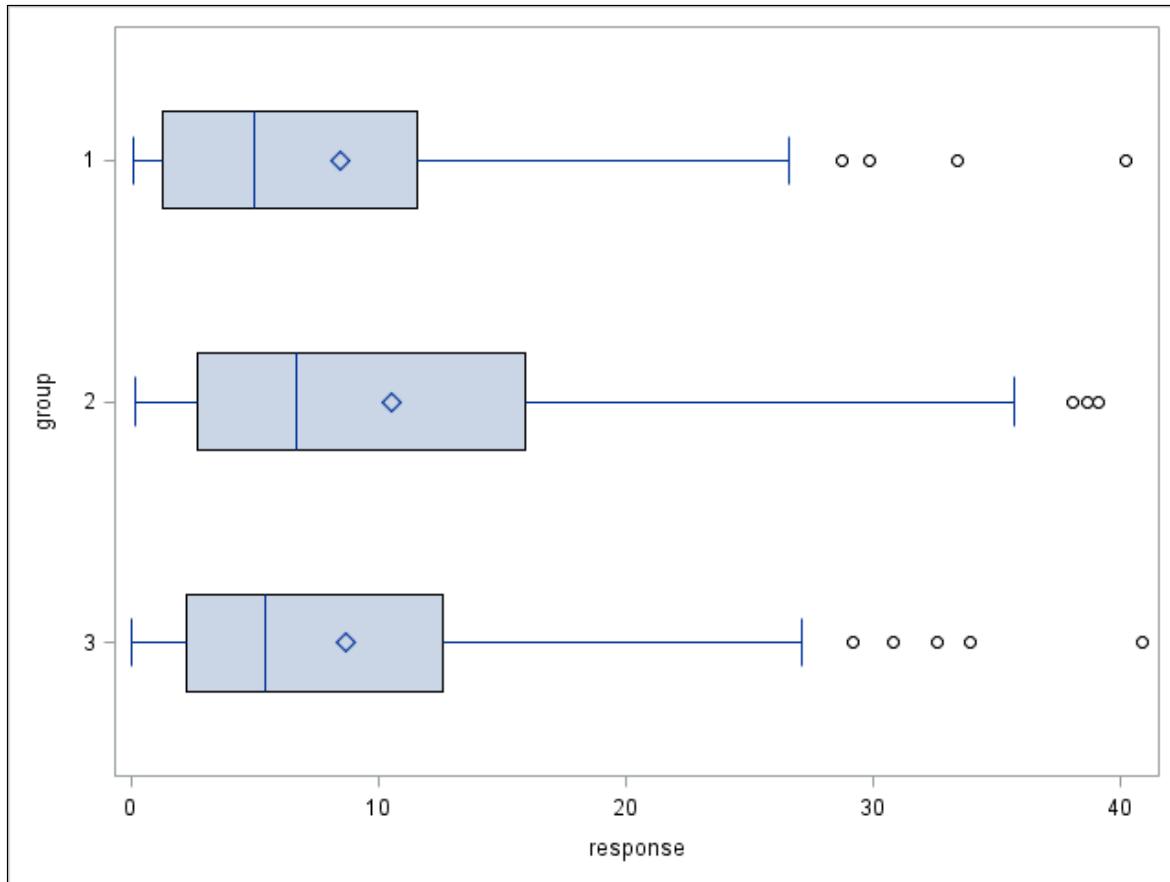
proc sgplot data=sp4r.boxplot_data;
  hbox response;
run;

proc sgplot data=sp4r.boxplot_data;
  hbox response / category=group;
run;
```

Selected PROC SGPlot statement and option:

HBOX creates a horizontal box plot of the response variable. Use the **CATEGORY=** option to create a box plot for each category of a classification variable.





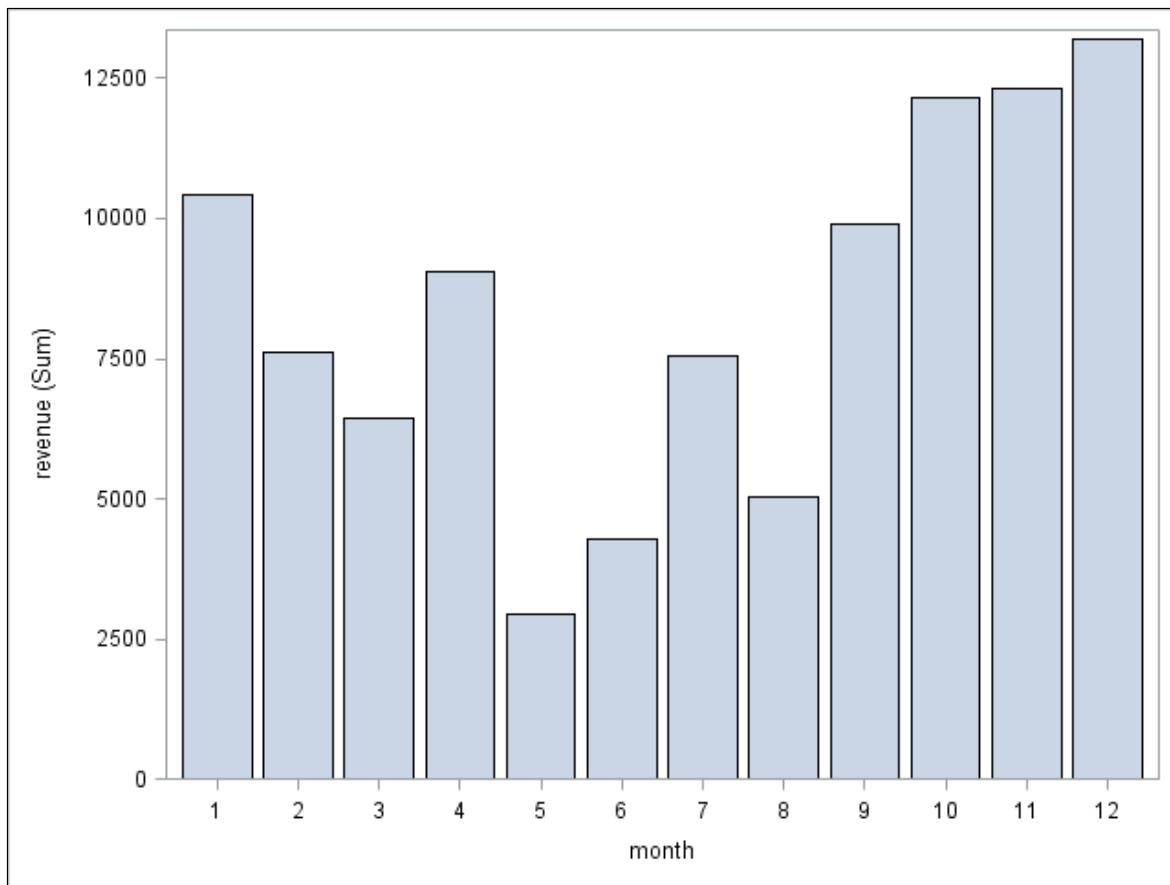
3. Create a data table with random deviates generated from a normal distribution with a mean of 10,000 and a standard deviation of 5000. This corresponds to the revenue for a 12-month period. Create a bar chart of the newly generated data.

```
data sp4r.sales;
  call streaminit(123);
  do month=1 to 12;
    revenue = rand('Normal',10000,5000);
    output;
  end;
run;

proc sgplot data=sp4r.sales;
  vbar month / response=revenue;
run;
```

Selected PROC SGPlot statement and option:

VBAR creates a vertical bar chart. The proceeding variable is the category. Use the RESPONSE= option to specify the response for each category to be plotted.



4. Create a data table with $Y_1 = 10 + X + \varepsilon$ and $Y_2 = 35 + \frac{X}{2} + \varepsilon$ where X ranges from 1 to 30 and $\varepsilon \sim N(0, \sigma = 5)$. Keep only the variables X , Y_1 , and Y_2 . Use PROC SGPlot to create a scatter plot and a series plot separately for both variables Y_1 and Y_2 . Then merge the two plots by providing multiple SCATTER and SERIES statements to a single SGPlot procedure.

```

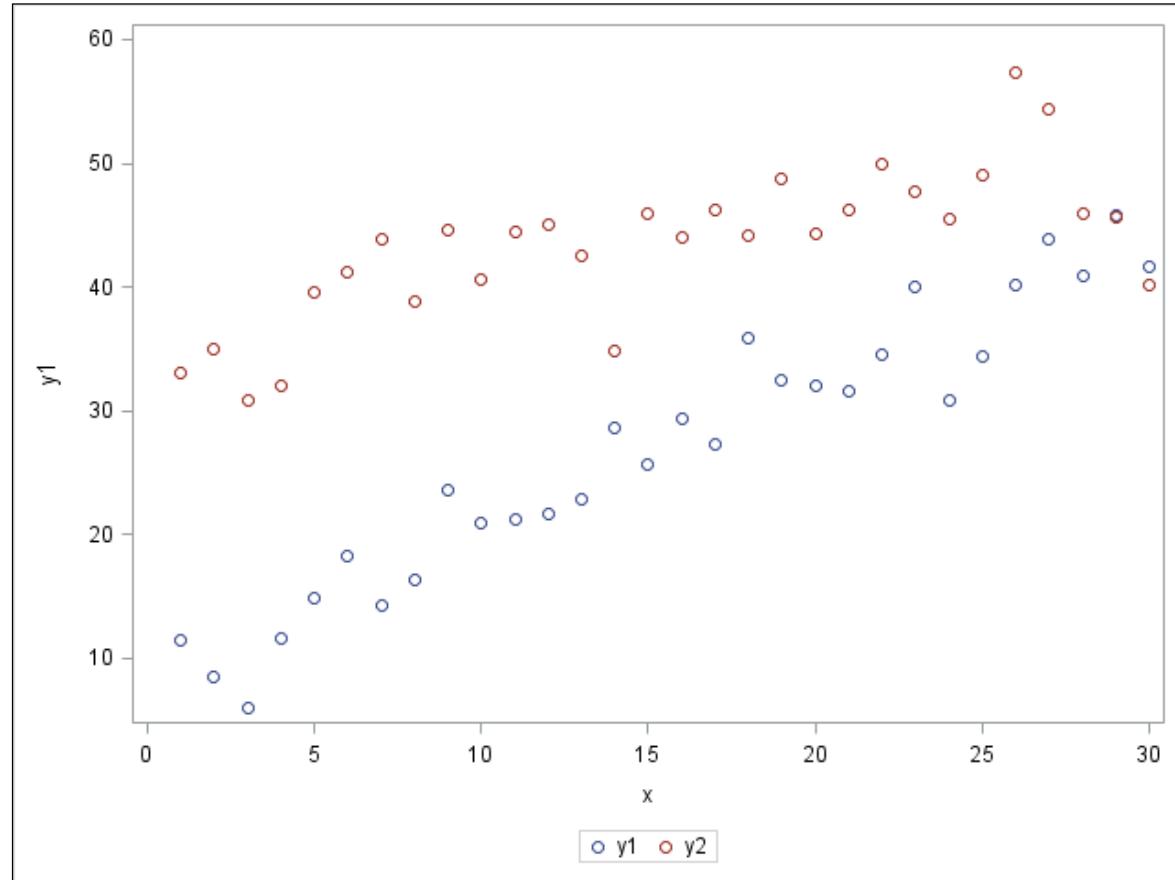
data sp4r.series_data (keep=x y1 y2);
  call streaminit(123);
  do x=1 to 30;
    beta01 = 10;
    beta11 = 1;
    y1 = beta01 + beta11*x + rand('Normal', 0, 5);
    beta02 = 35;
    beta12 = .5;
    y2 = beta02 + beta12*x + rand('Normal', 0, 5);
    output;
  end;
run;

proc sgplot data=sp4r.series_data;
  scatter x=x y=y1;
  scatter x=x y=y2;
run;
proc sgplot data=sp4r.series_data;

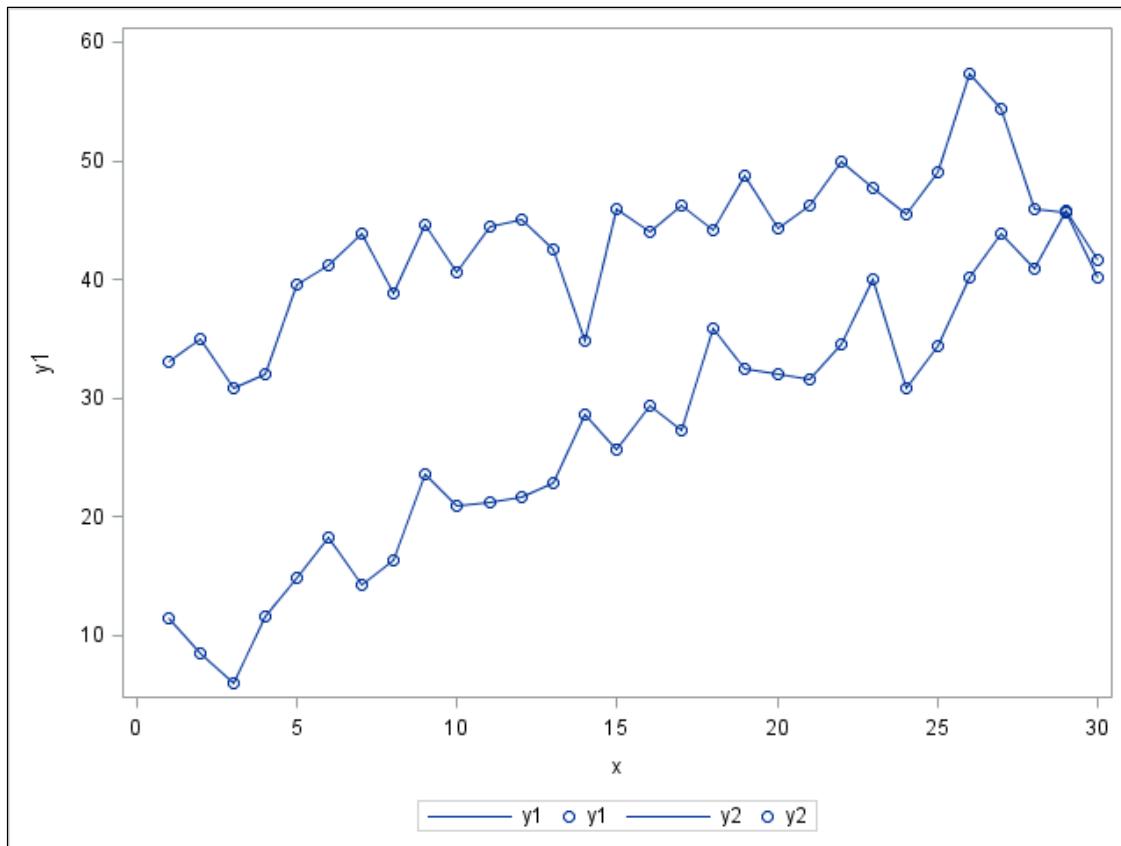
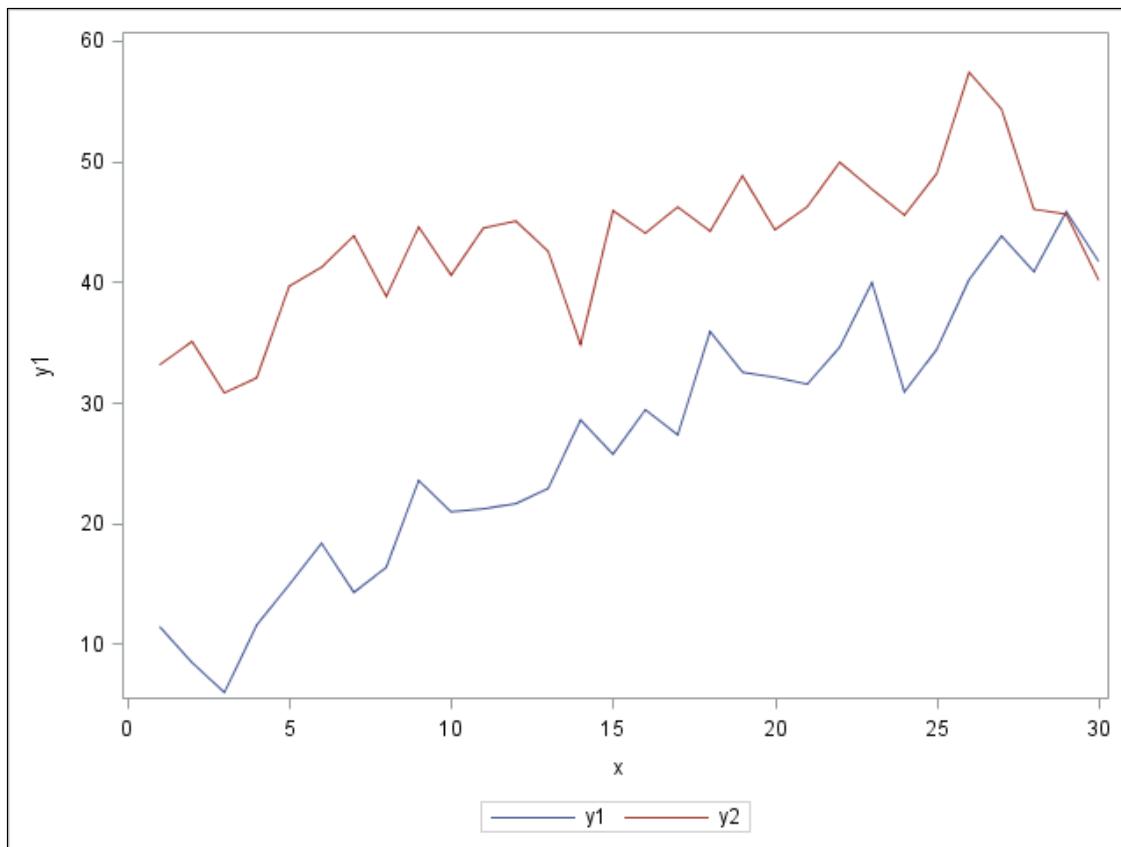
```

```
series x=x y=y1;
series x=x y=y2;
run;

proc sgplot data=sp4r.series_data;
series x=x y=y1;
scatter x=x y=y1;
series x=x y=y2;
scatter x=x y=y2;
run;
```



Notice that the Y-axis label uses the first Y-axis variable input name. This can be altered using the YAXIS LABEL= option.



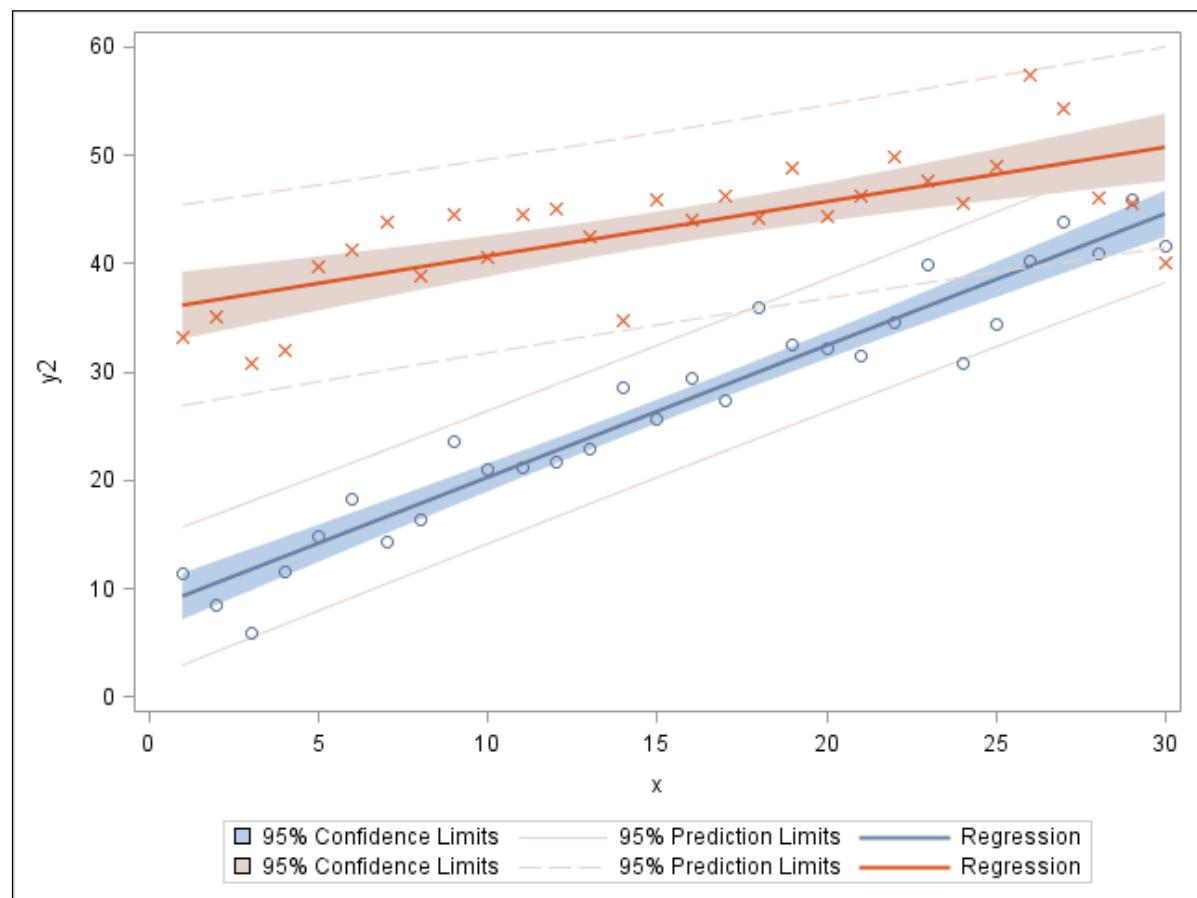
Notice that the legend specifies a name for both SCATTER and SERIES statements. This can be altered by using the KEYLEGEND statement and the NAME option.

- Using the data table from step 4, add a line of best fit to the scatter plot for both Y_1 and Y_2 .

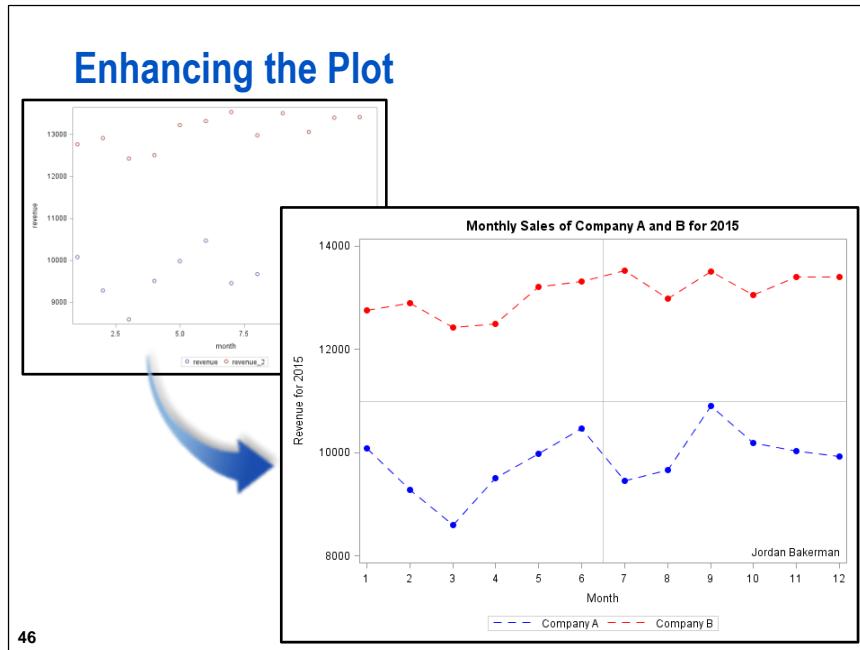
```
proc sgplot data=sp4r.series_data;
  reg x=x y=y1 / clm cli;
  reg x=x y=y2 / clm cli;
run;
```

Selected PROC SGPlot statement and options:

REG creates a fitted regression line to a scatter plot. Use the CLM and CLI options to include prediction and confidence limits. Limits corresponding to alpha=0.05 is the default.



End of Demonstration



Enhancing the Plot

Polish the plot with the following statements and options:

R	SAS Statements	SAS Options
Main	TITLE	
xlab ylab	XAXIS, YAXIS	LABEL=
xlim, ylim	XAXIS, YAXIS	MIN=, MAX=
axis	XAXIS, YAXIS	VALUES=()
type	SERIES, SCATTER	
col		COLOR=
lty	LINEATTRS=	PATTERN=
pch	MARKERATTRS=	SYMBOL=
abline	REFLINE	TRANSPARENCY=, AXIS=
legend	KEYLEGEND	LEGENDLABEL=, NAME=
text	INSET	POSITION=

47

Selected SAS statements and options:

TITLE specifies a title to be added to the plot followed by the string in quotation marks. Use the statements TITLE1, ..., TITLE10 to specify the line for each title. A maximum of 10 lines can be specified.

XAXIS specifies options for the x-axis. Use the LABEL= option to specify the X-axis label. Use the MIN= and MAX= options to specify the X-axis limits. Use the VALUES= option to specify the tick marks.

YAXIS	specifies options for the Y-axis. Use the LABEL= option to specify the Y-axis label. Use the MIN= and MAX= options to specify the Y-axis limits. Use the VALUES= option to specify the tick marks.
LINEATTRS	specifies a change in the line attributes. Use the COLOR= option to specify the color of the line and the PATTERN= option to specify the pattern of the line.
MARKERATTRS	specifies a change in the marker attributes. Use the COLOR= option to specify the color of the points and the SYMBOL= option to specify the symbol of the points.
REFLINE	creates a vertical or horizontal line within the plot. Use the AXIS= option to specify X for vertical and Y for horizontal. Use the TRANSPARENCY= [0,1] option to fade the line. A value of 1 provides the brightest line.
KEYLEGEND	adds a legend to the plot. Use the NAME= in the PLOT statement to identify which plots to include in the legend.
INSET	adds a text box inside the plot axes. Use the POSITION= option to specify the location.

Presenting the Plot

Present the plot by performing the following tasks:

- Copy and paste the plot into a Word document
- Use ODS to save the image.

```
ods pdf file = '&path\myplot.pdf';
      ODS PDF FILE='path\name.pdf';

proc sgplot data=sales;
  ...
run;

ods pdf close;
```

48

Easily copy and paste a plot into a Word document and resize it as needed. Use the ODS PDF FILE= statement to save a SAS plot as a PDF document. Use the PDF document to incorporate the plot into a LaTeX document.



Enhancing the Plot

SP4R04d03.sas

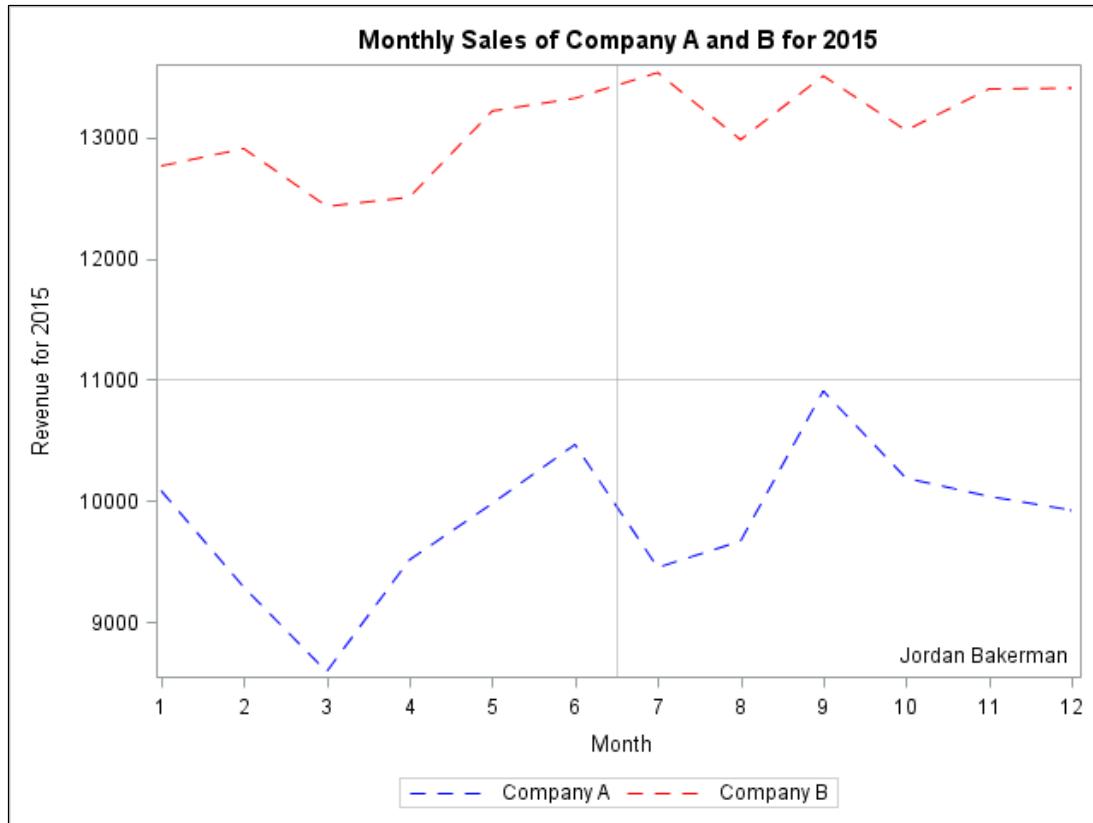
1. Create a data table called **Sales** with 12 observations corresponding to a 12-month period. In addition, create a variable called **Revenue**. This variable is randomly generated from a normal distribution with a mean of 10,000 and a standard deviation of 1,000. Create a second variable called **Revenue_2**. This variable is also randomly generated from a normal distribution with a mean of 13,000 and a standard deviation of 500.

```
data sp4r.sales;
  call streaminit(123);
  do month=1 to 12;
    revenue = rand('Normal',10000,1000);
    revenue_2 = rand('Normal',13000,500);
    output;
  end;
run;
```

2. Use PROC SGLOT to create a series plot with both variables.
 - a. Provide the title ‘Monthly Sales of Company A and B for 2015’, the X-axis label ‘Month’, the Y-axis label ‘Revenue for 2015’, a vertical reference line at the 6.5 month mark, and a horizontal reference line at 11,000.
 - b. Put your name in the bottom right corner with the INSET statement.
 - c. Provide tick marks from 1 to 12 by 1 for the X-axis.
 - d. Create the legend labels ‘Company A’ and ‘Company B’, with a blue line and a red line to distinguish them. Give each line a dashed pattern.

```
proc sgplot data=sp4r.sales;
  series x=month y=revenue / legendlabel='Company A'
    lineattrs=(color=blue pattern=dash);
  series x=month y=revenue_2 / legendlabel='Company B'
    lineattrs=(color=red pattern=dash);

  title 'Monthly Sales of Company A and B for 2015';
  xaxis label="Month" values=(1 to 12 by 1);
  yaxis label="Revenue for 2015";
  inset "Jordan Bakerman" / position=bottomright;
  refline 6.5 / transparency= 0.5 axis=x;
  refline 11000 / transparency= 0.5;
run;
```



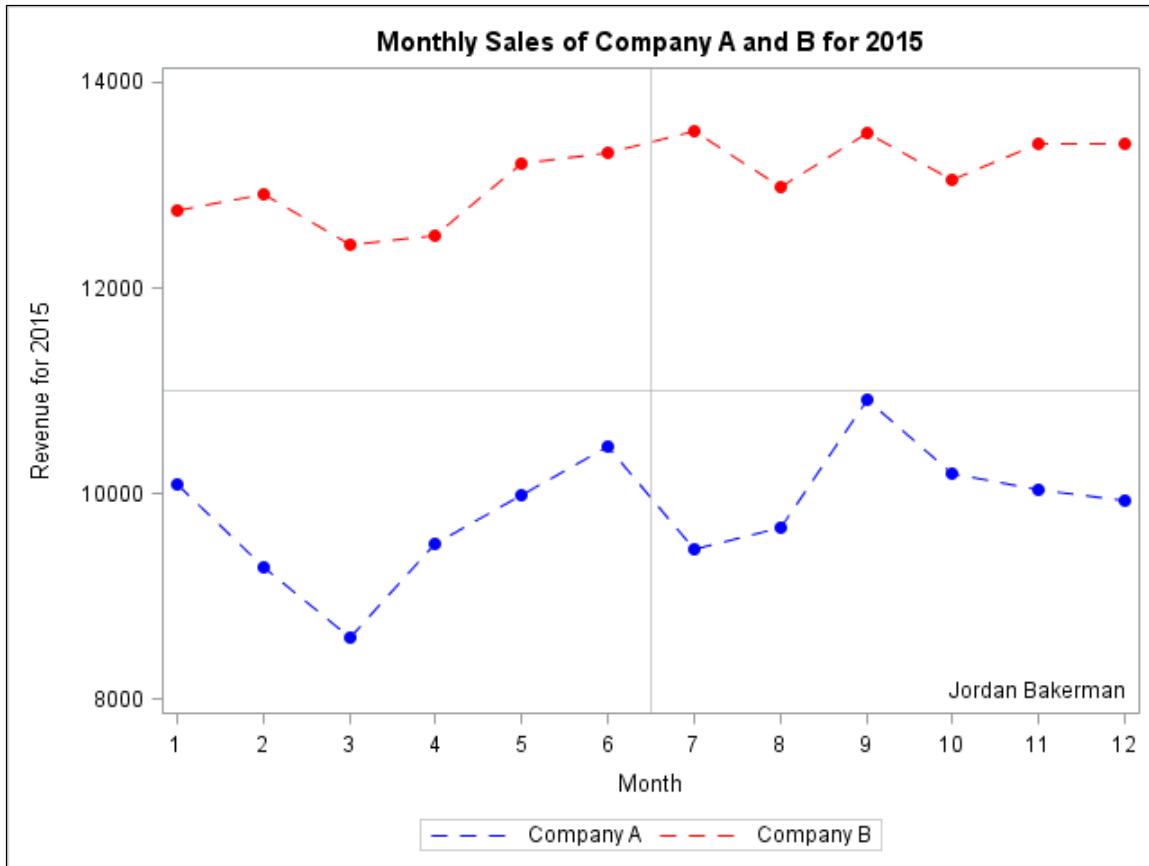
3. Use the **Sales** data table to further enhance the plot from the previous demonstration.
 - a. Use the SCATTER statement to add the observations to the plot.
 - b. Use the SYMBOL=circlefilled option for the points.
 - c. Add the KEYLEGEND statement to avoid multiple legends for both the SCATTER and SERIES statements.
 - d. Change the Y-axis limits to 8000 and 14000.

```

proc sgplot data=sp4r.sales;
  series x=month y=revenue / legendlabel='Company A' name='Company A'
    lineattrs=(color=blue pattern=dash);
  scatter x=month y=revenue / markerattrs=(color=blue
    symbol=circlefilled);
  series x=month y=revenue_2 / legendlabel='Company B'
    name='Company B' lineattrs=(color=red pattern=dash);
  scatter x=month y=revenue_2 / markerattrs=(color=red
    symbol=circlefilled);

  title 'Monthly Sales of Company A and B for 2015';
  xaxis label="Month" values=(1 to 12 by 1);
  yaxis label="Revenue for 2015" min=8000 max=14000;
  inset "Jordan Bakerman" / position=bottomright;
  refline 11000 / transparency= 0.5;
  refline 6.5 / transparency= 0.5 axis=x;
  keylegend 'Company A' 'Company B';
run;

```



End of Demonstration

4.3 Multi-Cell Plotting

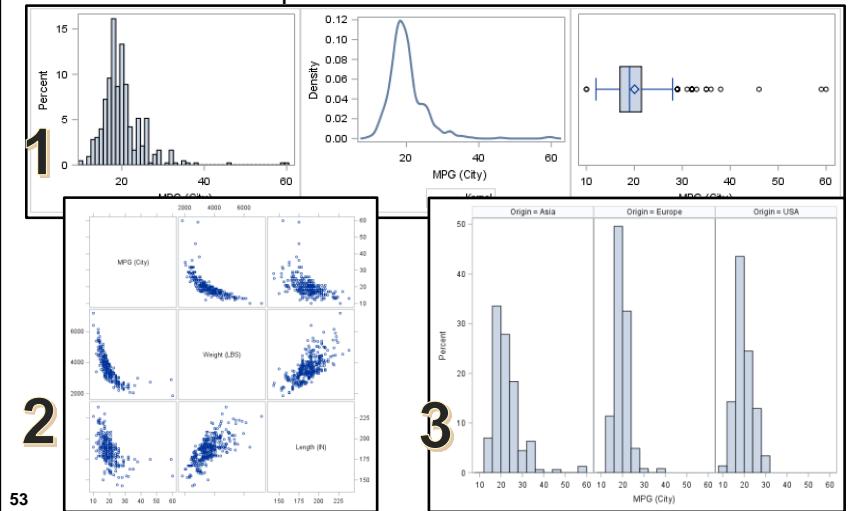
Objectives

- Use the SGSCATTER procedure to create scatter plot matrices and multi-cell scatter plots.
- Use the SGPANEL procedure to create a panel of plots for the values of one or more classification variables.
- Use the ODS LAYOUT statement to combine multiple plots of different value.

52

Motivation

Create multi-cell plots.



53

The first set of plots is an example of creating a diverse set of plots in the same window. The second set is a scatter plot matrix, and the third set is an example of creating a SAS panel of plots.

Duplicating the R Script

```
#Create a scatter plot matrix
n = 1000
x = rexp(n)
y = rnorm(n,3,1)
z = rchisq(n,10)
pairs(~x+y+z)

#Create side by side histograms
n=1000
fem = rnorm(n,66,2)
mal = rnorm(n,72,2)
par(mfrow=c(1,2))
hist(fem,50,main="Histogram of Female Heights")
hist(mal,50,main="Histogram of Female Heights")

#create a window with multiple plots
n=1000
fem = rnorm(n,66,2)
par(mfrow=c(1,3))
hist(fem,50,main="Histogram of Female Heights")
plot(density(fem),main="Density Estimate of Female Heights")
boxplot(fem,main="Boxplot of Female Heights")
```

54

SGSCATTER Procedure

The SGSCATTER procedure creates a paneled graph of scatter plots for multiple combinations of variables depending on the PLOT statement.

- MATRIX – creates a scatter plot matrix.
- PLOT – creates a paneled graph that contains multiple independent scatter plots.
- COMPARE – creates a comparative panel of scatter plots with shared axes.

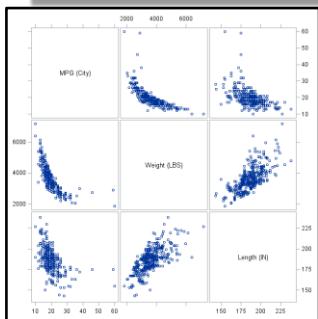
55

SGSCATTER Procedure

Use the MATRIX statement to duplicate the pairs() function in R.

```
proc sgscatter data=sp4r.cars;
  matrix mpg_city weight length;
run;
```

MATRIX variable-1 variable-2 ... </ options>;



56

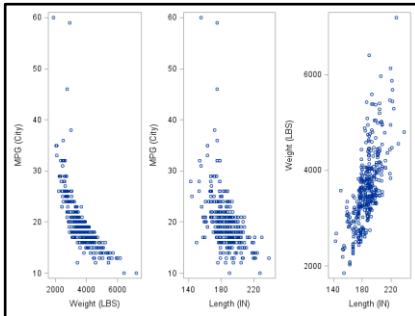
PROC SGSCATTER creates a scatter plot matrix of all the variables that are specified in the MATRIX statement. Options in the MATRIX statement enable both histograms and density estimates to be plotted on the diagonal of the scatter plot matrix.

SGSCATTER Procedure

Use the PLOT statement to create multi-cell scatter plots.

```
proc sgscatter data=sp4r.cars;
  plot mpg_city*weight mpg_city*length
    weight*length / columns=3;
run;
```

PLOT variable-i * variable-j ... </ options>;



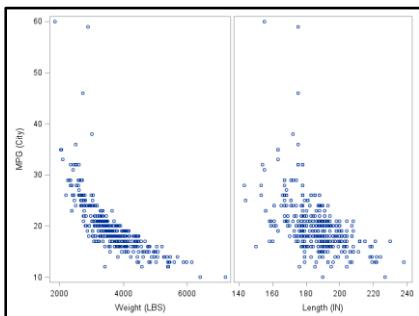
57

Create a combined window of scatter plots with PROC SGSCATTER and the PLOT statement. Provide the PLOT statement with the crossed variables to be plotted. Options include the ROWS= and COLUMNS= specifications, which enable the user to set the matrix structure.

SGSCATTER Procedure

Use the COMPARE statement to create a comparative panel of scatter plots with shared axes.

```
proc sgscatter data=sp4r.cars;
  compare y=(mpg_city) x=(weight length);
run;
COMPARE X=(variable-i...) Y=(variable-j...)... </ options>;
```



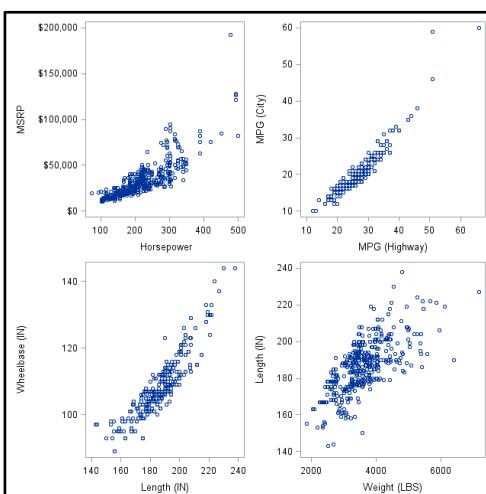
58

The COMPARE statement enables the user to create multiple plot with one or more shared axes. The dimension of the graph is determined by the number of variables in the Y= and X= statements.

4.05 Multiple Choice Poll

Which SGPlot procedure statement is used to create the plot?

- a. MATRIX
- b. PLOT
- c. COMPARE



59

Multi-Cell Plots

Use the ODS LAYOUT statement to reproduce the `par(mfrow=c(),)` function in R.

```
ods layout start rows=1 columns=3;

ODS LAYOUT START ROWS= COLUMNS=
  <WIDTH= HEIGHT=
  ROW_HEIGHT= COLUMN_HEIGHT=
  ROW_GUTTER= COLUMN_GUTTER=
  options>;
...
ODS LAYOUT END;

ods layout end;
```

61

To combine plots of different types, use the ODS LAYOUT statement. Each ODS LAYOUT statement should end with ODS LAYOUT END to revert to the default plotting layout.

Selected ODS LAYOUT options:

- ROW=** specifies the number of rows for the layout.
- COLUMNS=** specifies the number of columns for the layout.
- WIDTH=** specifies the width dimension of each plot.
- HEIGHT=** specifies the height dimension of each plot.
- ROW_HEIGHT=** specifies the height of each row. The user can provide a list.
- COLUMN_HEIGHT=** specifies the height of each column. The user can provide a list.
- ROW_GUTTER=** specifies the distance between rows of the layout.
- COLUMN_GUTTER=** specifies the distance between columns of the layout.

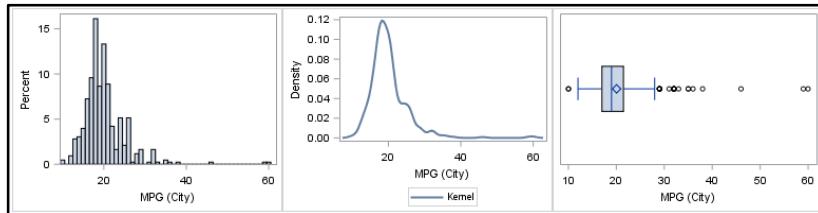
Valid units of measure are (cm) and (in).

Multi-Cell Plots

Use the ODS REGION statement to specify the location of each plot.

```
ods region row=1 column=3;
proc sgplot data=sp4r.cars;
  hbox mpg_city;
run;
```

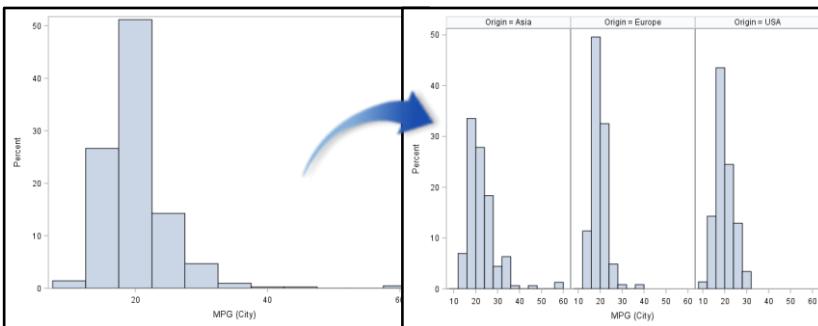
ODS REGION ROW= COLUMN=;



62

SGPANEL Procedure

Use the SGPANEL procedure to create a panel of plots according to one or more classification variables.



63

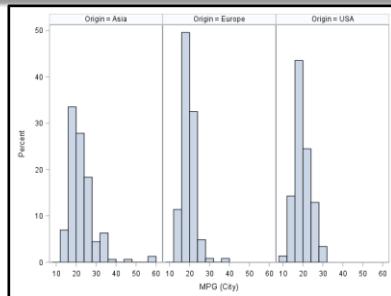
The SGPANEL procedure combines plots of the same type only. The panel automatically generates a title for each plot according to the classification variable.

The SG PANEL Procedure

Use the PANELBY statement to specify the classification variable.

```
proc sgpanel data=sp4r.cars;
  panelby origin / columns=3;
  histogram mpg_city; PANELBY classification-variable;
run;
```

PLOTNAME response-or-category-variable </ options>;



64

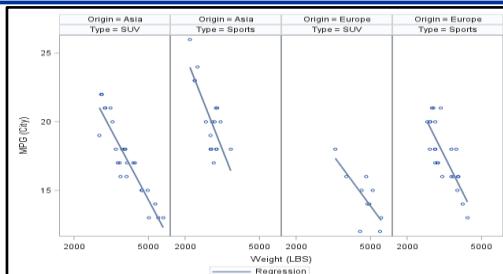
PROC SG PANEL is followed by the PANELBY statement, which enables the user to specify a classification variable. The panel creates the same plot type for each classification and response. All the plot types from the “Single-Cell Plotting” section can be used with PROC SG PANEL.

SGPANEL Procedure

The SGPANEL procedure supports

- multiple Classification variables
- the same PLOT statements from PROC SG PLOT.

```
proc sgpanel data=sp4r.lesscars;
  panelby origin type / rows=1 columns=4;
  reg x=weight y=mpg_city;
run;
```



65

Use the ROWS= and the COLUMNS= options to specify the structure of the panel.

The lattice layout supports the use of exactly two classification variables.



Multi-Cell Plotting

SP4R04d04.sas

1. Create a data table with the variable **Sex** that has groups *F* and *M*. For each group, generate 1000 observations with a seed of 123. If **Sex** is *F*, let **Height** be normally distributed with a mean of 66 and a standard deviation of 2. If **Sex** is *M*, let **Height** be normally distributed with a mean of 72 and a standard deviation of 2.

```
data sp4r.multi;
  call streaminit(123);
  do Sex='F', 'M';
    do j=1 to 1000;
      if sex='F' then height = rand('Normal',66,2);
      else height = rand('Normal',72,2);
      output;
    end;
  end;
run;
```

The DO loop can also iterate over a list of character variables. For each iteration, the variable is filled with the specified name. The DO loop does not require the TO syntax when a list is provided.

A comma is needed for any list, character, or numeric values.

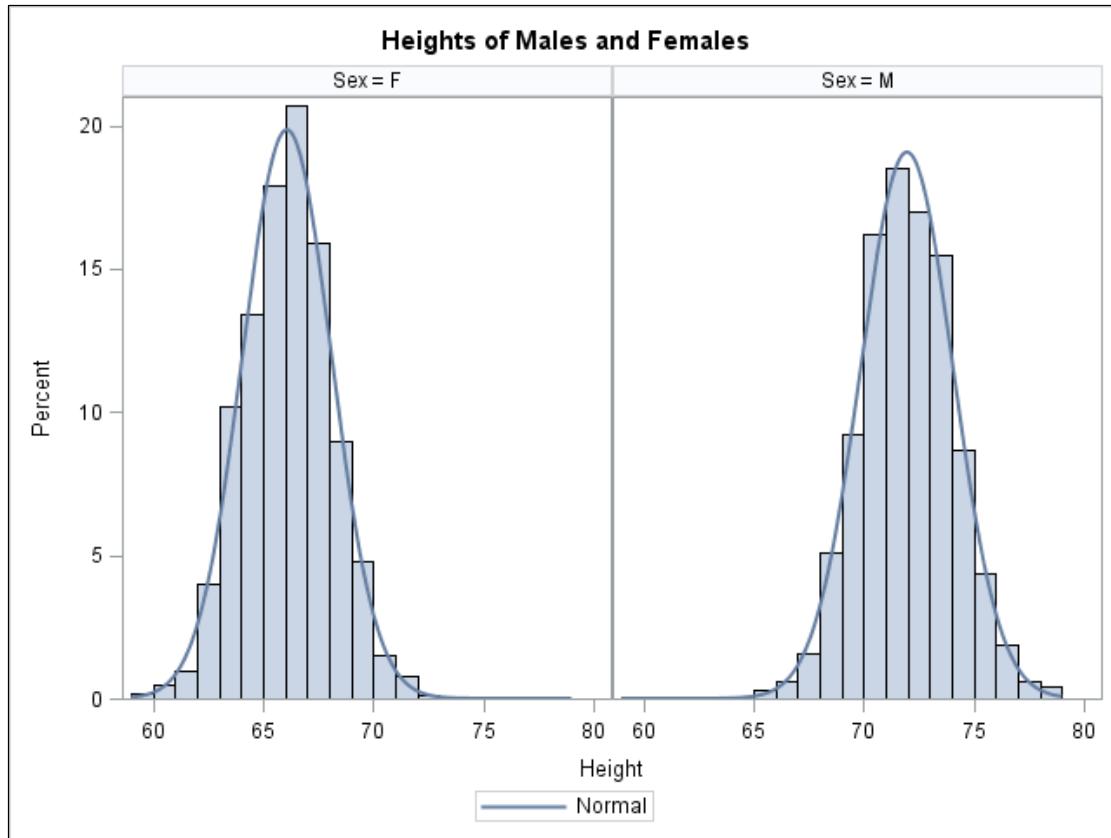
2. Use the SGPALEL procedure to plot histograms for each sex.
 - a. Overlay a normal density estimate as well.
 - b. Provide the title ‘Heights of Males and Females’ and the X-axis title ‘Height’.

```
proc sgpanel data=sp4r.multi;
  panelby sex;
  histogram height;
  density height / type=normal;
  title 'Heights of Males and Females';
  colaxis label='Height';
run;
```

Selected PROC SGPALEL statements:

PANELBY creates a plot for each category of the proceeding variable and combines the plots in a single panel.

COLAXIS allows for the specification of the Y-axis options. The YAXIS statement is not used in the SGPALEL procedure.



3. Using the same data table, create three different plots side by side. Use the ODS LAYOUT statement.
 - a. Create a template with one row and three columns
 - b. Specify a height and width of three inches for each plot.
 - c. Use a WHERE statement to plot a histogram, a density estimate plot, and a box plot for Females only.
 - d. Provide appropriate titles for each plot.

```
ods layout Start rows=1 columns=3 row_height=(1in) column_gutter=0;

ods region row=1 column=1;
proc sgplot data=sp4r.multi (where= (sex='F'));
  histogram height / binwidth=.5;
  title 'Histogram of Female Heights';
run;

ods region row=1 column=2;
proc sgplot data=sp4r.multi (where= (sex='F'));
  density height / type=kernel;
  title 'Density Estimate of Female Heights';
run;
```

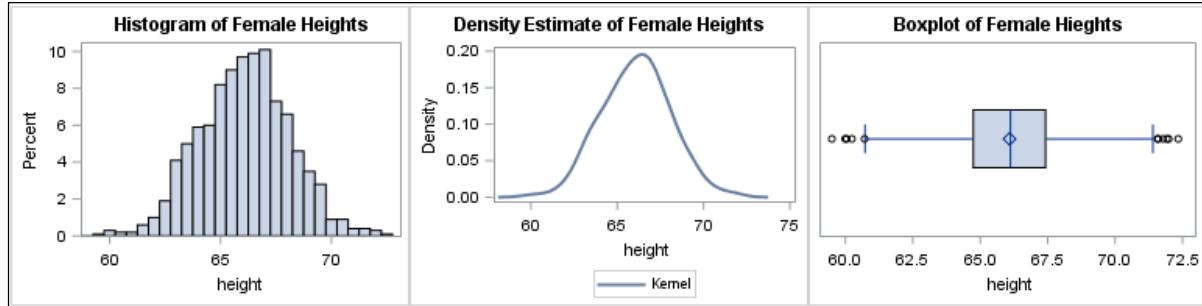
```

ods region row=1 column=3;
proc sgplot data=sp4r.multi (where= (sex='F'));
    hbox height;
    title 'Boxplot of Female Heights';
run;

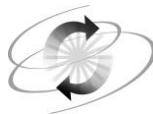
ods layout end;

```

The WHERE option is specified in parentheses inside the DATA statement. This is useful for plotting only a subset of data.



End of Demonstration



Exercises

1. Using the DO Loop and Creating Random Data Sets

- Navigate to the **SAS RAND function** page and choose a few functions to practice generating random numbers. Create a data table with at least two variables of random numbers and at least 10 observations. Be sure to use a random seed of your choice.
<http://support.sas.com/documentation/cdl/en/lefunctionsref/67960/HTML/default/viewer.htm#p0fpeei0opypg8n1b06qe4r040lv.htm>
- Create a new data table with the same random variables that you specified from the previous step. Create a variable called **Class** that groups the first five observations into class 1 and the second five into class 2. Drop the nested DO loop index variable from the data table and add a sequence from 1 to 10. Print the data upon completion.
- Run the SAS code below (**SP4R04e02.sas**). What do you notice?

```

data test;
  do i=1 to 2;
    output;
  end;
run;

proc print data=test;
run;

data test;
  set test;
  do j=1 to 5;
    output;
  end;
run;

proc print data=test;
run;

```

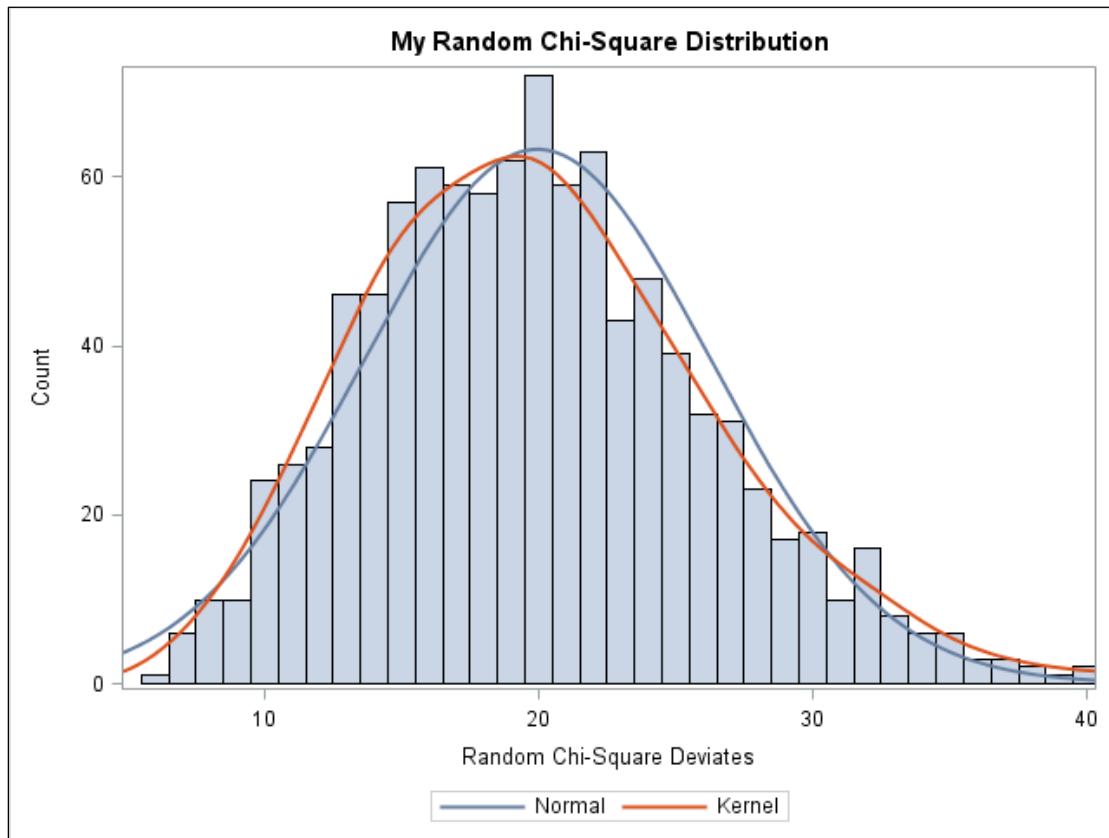
2. Exploring PDF, CDF, and Quantiles Variables

- Use the DO loop to create quantiles from 0 to 10 by 1.
- Identify the density and the cumulative density of a binomial distribution with parameters 0.8 and 10 by creating the variables **PDF** and **CDF**.
- Use the **CDF** variable to create the variable **Quantile**, which mirrors the DO loop values.
- Print the data upon completion.

3. Plotting Chi-Square Random Numbers

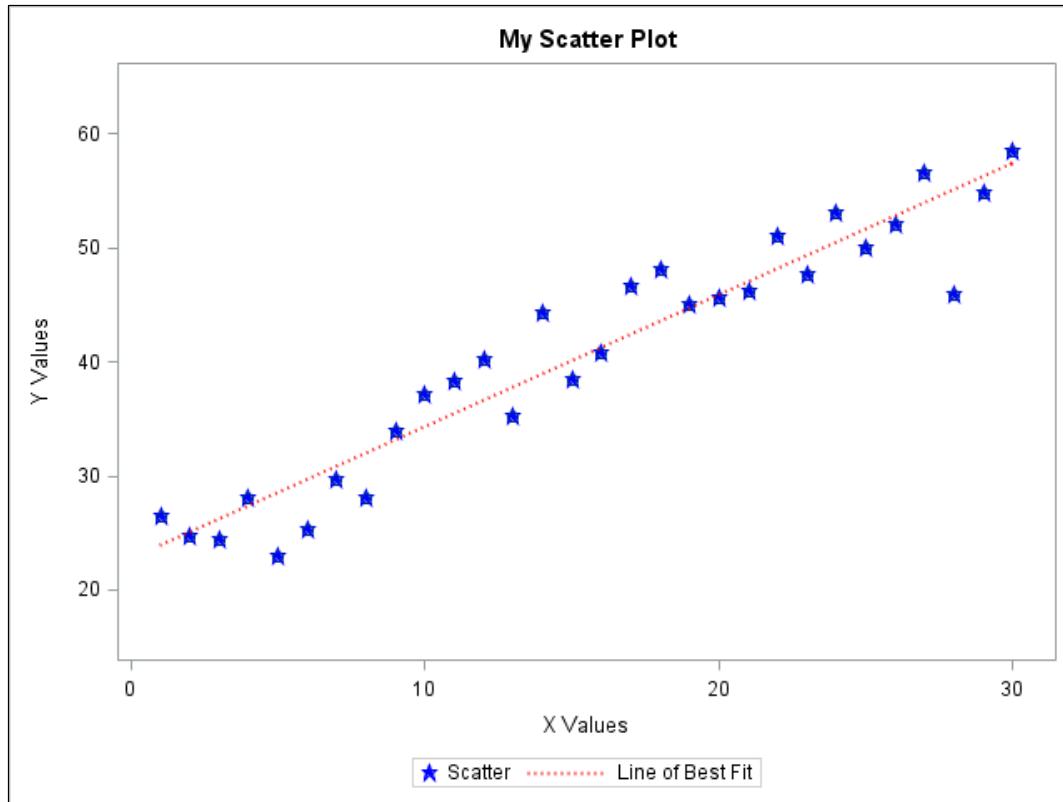
- Create a data table with 1000 random deviates from a chi-square distribution with 20 degrees of freedom and a seed of 123.

- b. Use PROC SGPlot to plot a histogram of the data.
- 1) Alter the appearance of the plot by setting the BINWIDTH= option to 1.
 - 2) Add both a normal and kernel density estimate.
 - 3) Add the title ‘My Random Chi-Square Distribution’.
 - 4) Add the X-axis title ‘Random Chi-Square Deviates’.
 - 5) Use X-axis limits of 5 and 40.
 - 6) Request the frequency instead of the percent by providing the option SCALE=COUNT in the HISTOGRAM statement.

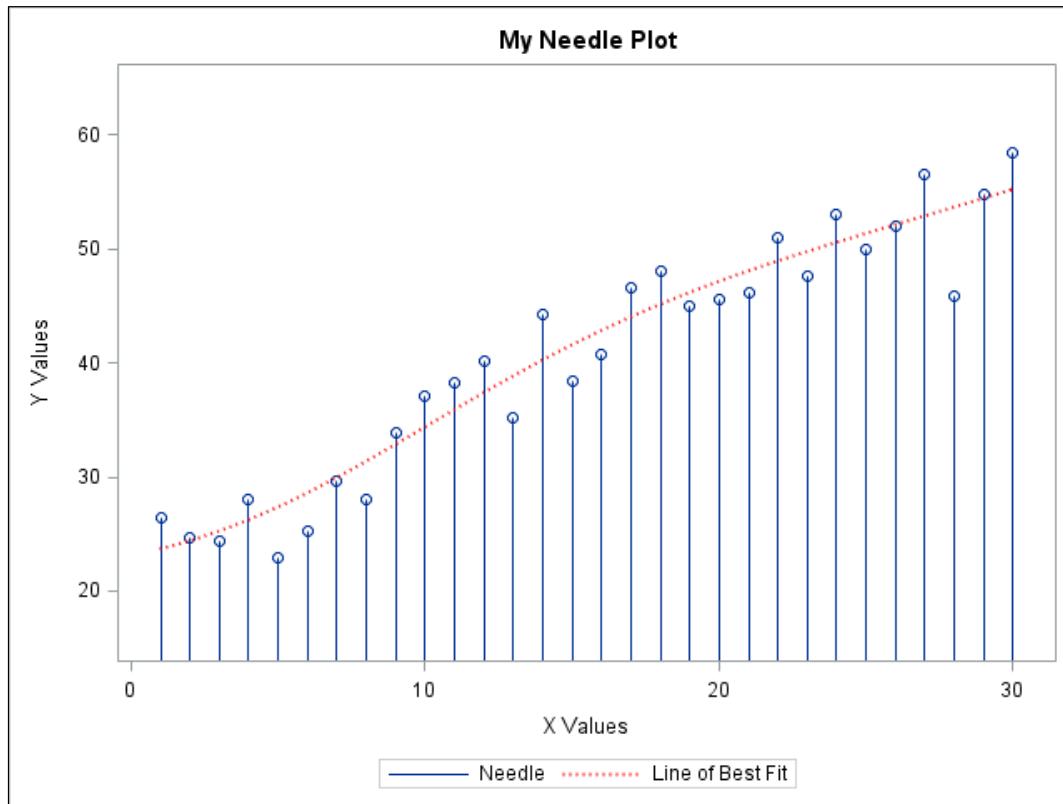


4. Plotting Simple Linear Regression Data

- a. Create a data table with $Y = \beta_0 + \beta_1 X + \varepsilon$ where X ranges from 1 to 30, $\beta_0 = 25$, $\beta_1 = 1$, and $\varepsilon \sim N(\mu = 0, \sigma = 5)$. Keep only the variables X and Y .
- b. Use PROC SGPlot and the REG statement to plot the line of best fit for the data. Create a plot of the data. Use both the SCATTER and REG statement to plot the points and a line of best fit.
 - 1) Enhance the plot by coloring the points blue and using the symbol STARFILLED
 - 2) Color the regression line red and use the pattern DOT.
 - 3) Add a title of your choosing to the X-axis, Y-axis, and the main title.
 - 4) Use the X-axis limits from 0 to 31, and the Y-axis limits from 15 to 65.
 - 5) Name the legend ‘Scatter’ and ‘Line of Best Fit’ for both plot types.

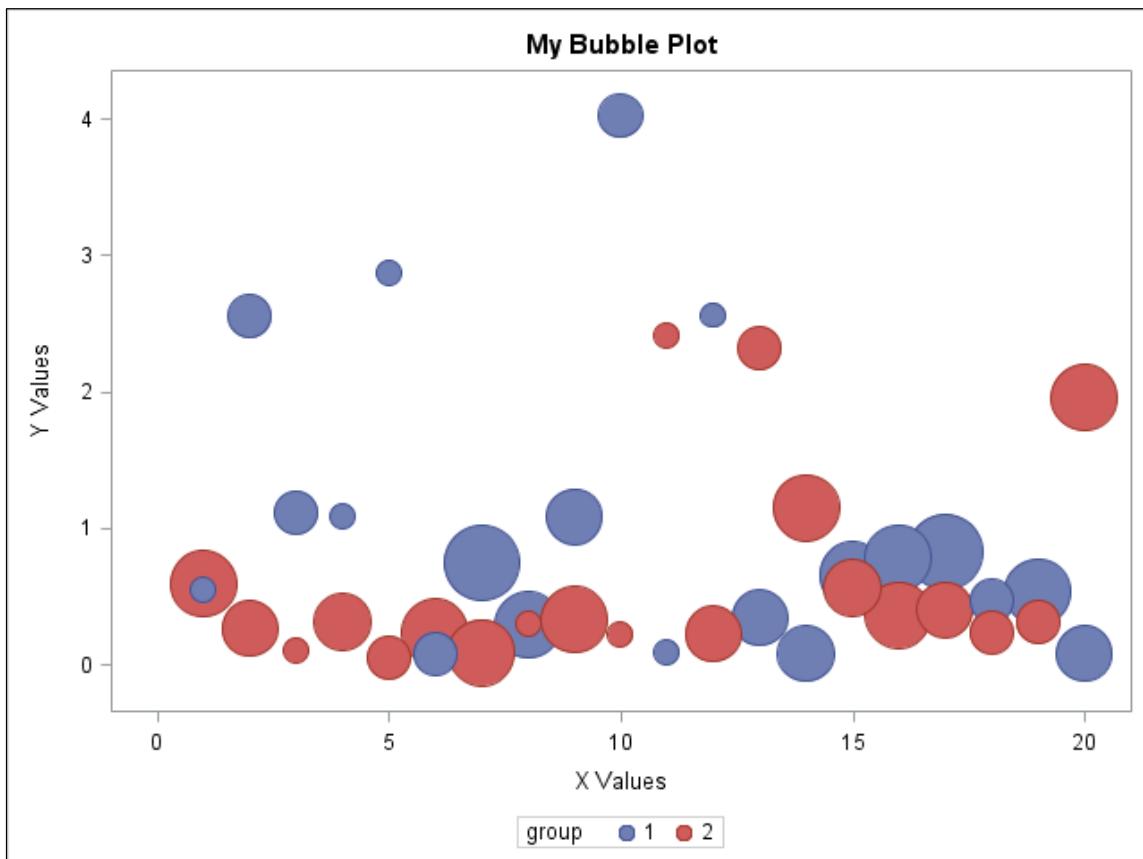


- c. Alter the previous plot by changing the SCATTER statement to NEEDLE and the REG statement to PBSPLINE. (This demonstrates the ease in which plot types can be altered.)



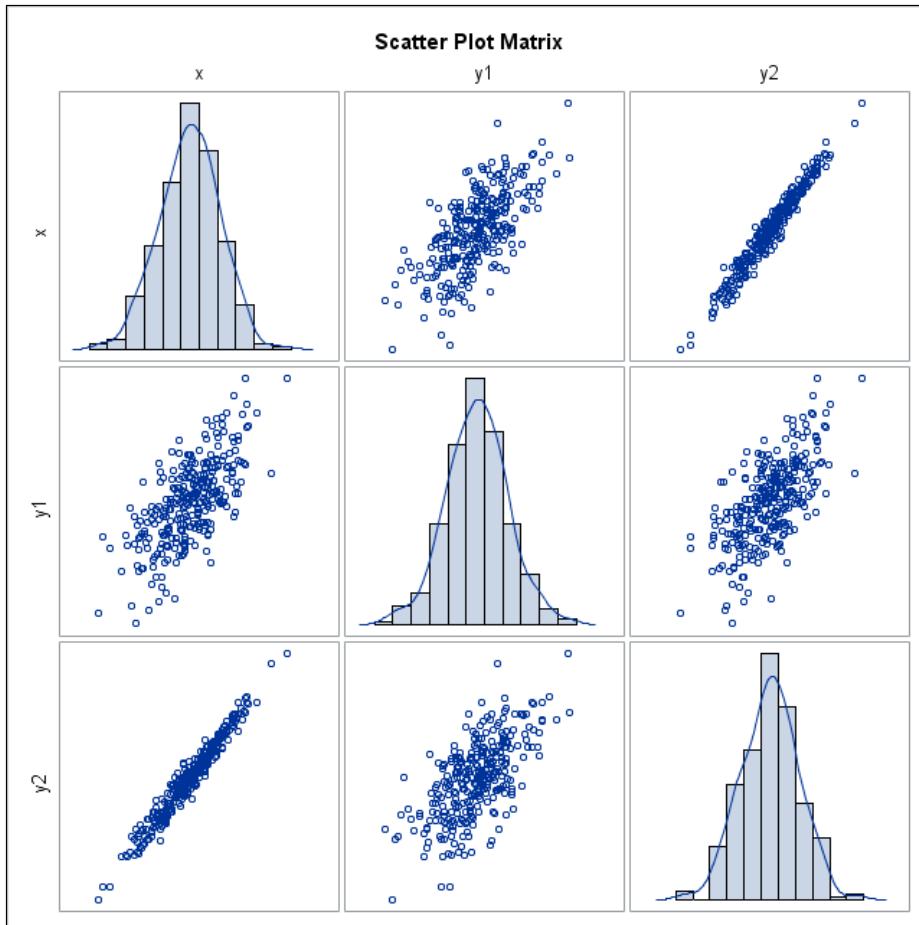
5. Creating a Bubble Plot

- Create a data table with two groups of 20 and the random seed 123. Create two random variables. Let the first be exponential and the second be binomial with parameters 0.5 and 5.
- Use the BUBBLE statement to create a bubble plot. Set the SIZE= to the binomial random variable. Also, specify the GROUP= option based on the two separate groups. Finally, provide the plot with titles for the X-axis, Y-axis, and main title.

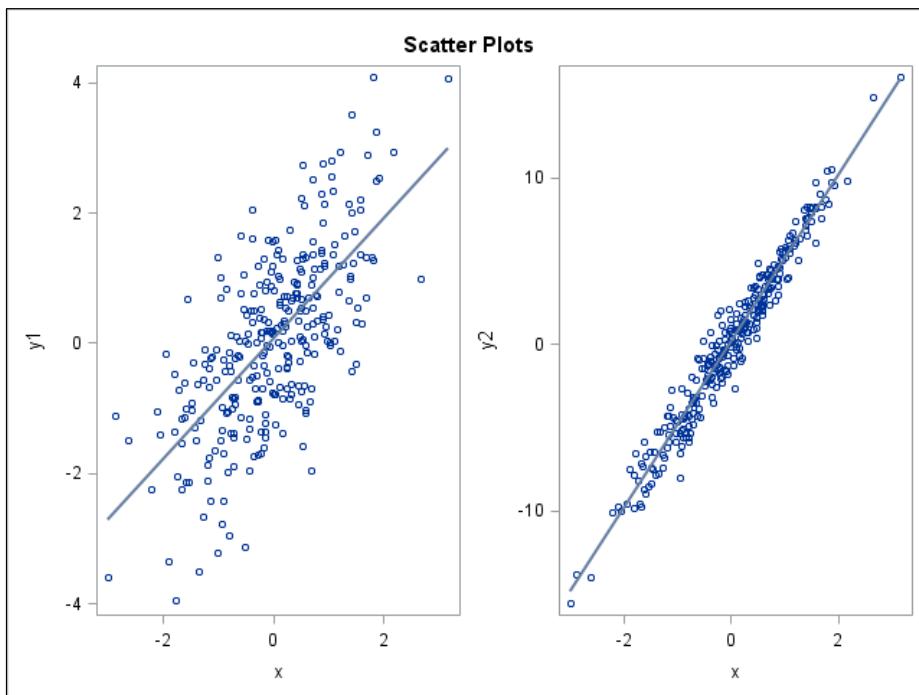


6. Using PROC SGSCATTER

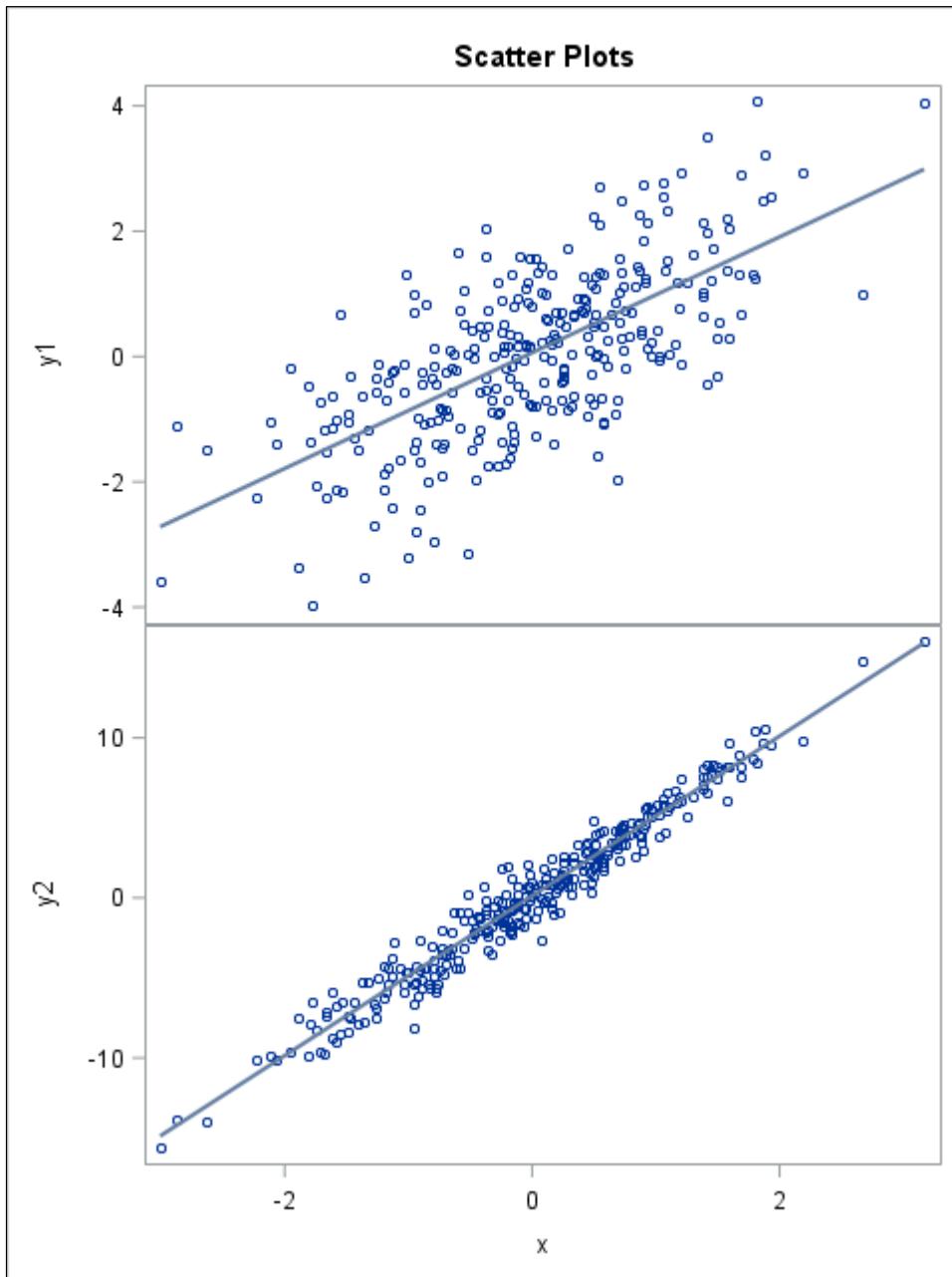
- Create a data table with 300 observations and a seed of 123.
 - Let **X** be the deviates from the standard normal distribution.
 - Produce a variable **Y1**, which is **X** plus standard normal deviates.
 - Produce another variable such that **Y2** is $5 \times \mathbf{X}$ plus standard normal deviates.
- Use PROC SGSCATTER to create a scatter plot matrix of **X**, **Y1**, and **Y2**. Include histograms and kernel density estimates on the diagonal. (Hint: Look up the **DIAGONAL=** option in the **MATRIX** statement of the SGSCATTER procedure.)



- c. Use PROC SGSCATTER to create side-by-side scatter plots of Y1 by X and Y2 by X with the PLOT statement. Add the regression line to both plots with the REG option.



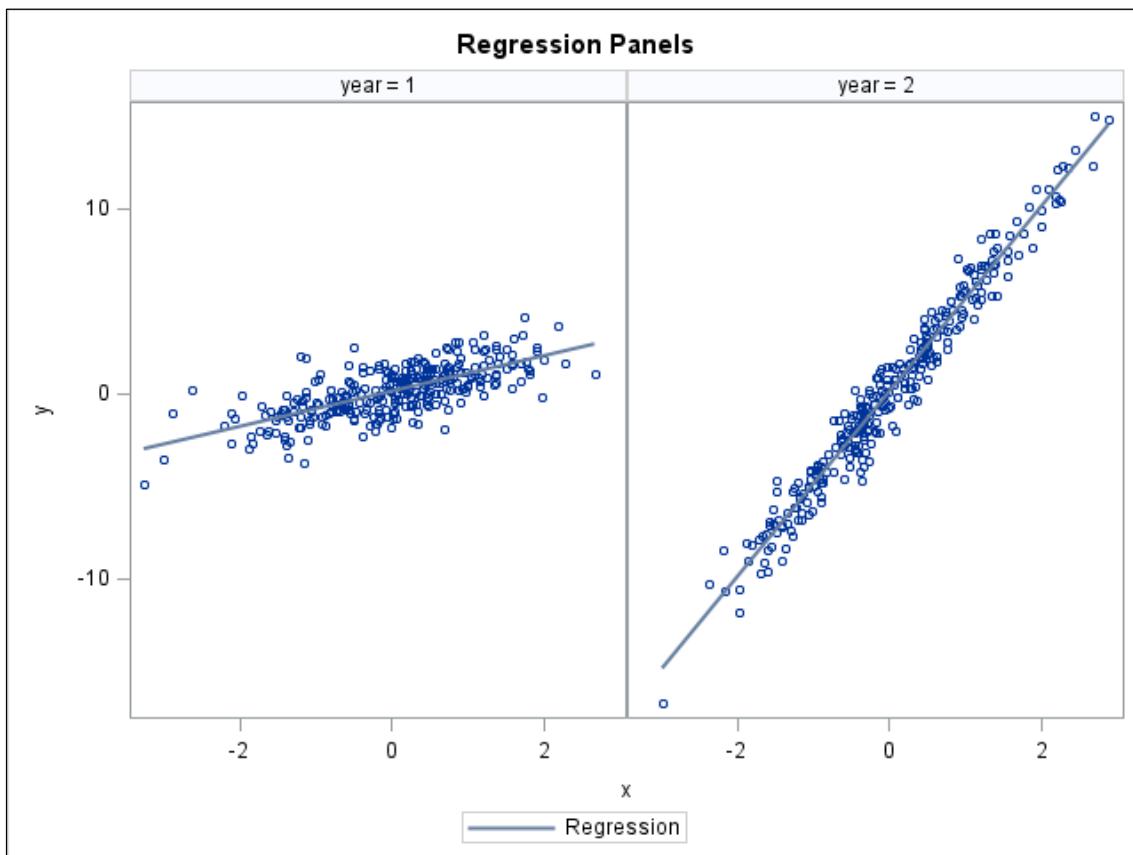
- d. Use PROC SGSCATTER and the COMPARE statement to create the same scatter plot with shared axes.



7. Using PROC SGPANEL

- a. Instead of creating **Y1** and **Y2** as separate variables as in the previous exercise, stack the variables in a single column denoted **Y** using a nested DO loop.
- 1) Create a categorical variable called **Year** with groups 1 and 2.
 - 2) Generate 300 observations for each group with a random seed of 123.

- 3) Let \mathbf{X} be the deviates from a standard normal distribution.
 - 4) Use IF-THEN/ELSE syntax to let \mathbf{Y} be \mathbf{X} plus standard normal deviates if **Year** is 1 and let \mathbf{Y} be $5 \cdot \mathbf{X}$ plus standard normal deviates otherwise.
- b. Use PROC SGPANEL to create a regression panel by year.



End of Exercises

4.4 Solutions

Solutions to Exercises

1. Using the DO loop and Creating Random Data Sets

- a. Navigate to the **SAS RAND function** page and choose a few functions to practice generating random numbers. Create a data table with at least two variables of random numbers and at least 10 observations. Be sure to use a random seed of your choice.

```
data sp4r.random;
  call streaminit(123);
  do i=1 to 10;
    rt = rand('T',5);
    rf = rand('F',3,4);
    ru = int(rand('Uniform')*10);
    output;
  end;
run;

proc print data=sp4r.random;
run;
```

Obs	i	rt	rf	ru
1	1	0.15554	0.57611	3
2	2	-0.71020	0.15053	2
3	3	-0.02583	0.04516	9
4	4	0.73364	0.25261	7
5	5	0.18336	0.88293	4
6	6	0.13730	1.50425	9
7	7	0.90893	2.18254	9
8	8	0.04611	0.10342	8
9	9	2.41523	0.55436	5
10	10	0.20044	1.59396	1

- b. Create a new data table with the same random variables that you specified from the previous step. Create a variable called **Class** that groups the first five observations into class 1 and the second five into class 2. Drop the nested DO loop index variable from the data table and add a sequence from 1 to 10. Print the data upon completion.

```
data sp4r.random (drop=j);
  call streaminit(123);
  do class=1 to 2;
    do j=1 to 5;
      sequence + 1;
      rt = rand('T',5);
      rf = rand('F',3,4);
      ru = int(rand('Uniform')*10);
      output;
    end;
  end;
run;

proc print data=sp4r.random;
run;
```

Obs	class	sequence	rt	rf	ru
1	1	1	0.15554	0.57611	3
2	1	2	-0.71020	0.15053	2
3	1	3	-0.02583	0.04516	9
4	1	4	0.73364	0.25261	7
5	1	5	0.18336	0.88293	4
6	2	6	0.13730	1.50425	9
7	2	7	0.90893	2.18254	9
8	2	8	0.04611	0.10342	8
9	2	9	2.41523	0.55436	5
10	2	10	0.20044	1.59396	1

- c. Run the SAS code below. What do you notice?

```

data test;
  do i=1 to 2;
    output;
  end;
run;

proc print data=test;
run;

data test;
  set test;
  do j=1 to 5;
    output;
  end;
run;

proc print data=test;
run;

```

Obs	i
1	1
2	2

Obs	i	j
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	2	1
7	2	2
8	2	3
9	2	4
10	2	5

The loop iterates through each observation in the data table.

2. Exploring PDF, CDF, and Quantiles Variables

- Use the DO loop to create quantiles from 0 to 10 by 1.
- Identify the density and the cumulative density of a binomial distribution with parameters 0.8 and 10 by creating variables **PDF** and **CDF**.
- Use the **CDF** variable to create the variable **Quantile**, which mirrors the DO loop values.
- Print the data upon completion.

```
data sp4r.random;
do q=0 to 10 by 1;
  pdf = pdf('Binomial',q,.8,10);
  cdf = cdf('Binomial',q,.8,10);
  quantile = quantile('Binomial',cdf,.8,10);
  output;
end;
run;

proc print data=sp4r.random;
run;
```

	Obs	q	pdf	cdf	quantile
	1	0	0.00000	0.00000	0
	2	1	0.00000	0.00000	1
	3	2	0.00007	0.00008	2
	4	3	0.00079	0.00086	3
	5	4	0.00551	0.00637	4
	6	5	0.02642	0.03279	5
	7	6	0.08808	0.12087	6
	8	7	0.20133	0.32220	7
	9	8	0.30199	0.62419	8
	10	9	0.26844	0.89263	9
	11	10	0.10737	1.00000	10

3. Plotting Chi-Square Random Numbers

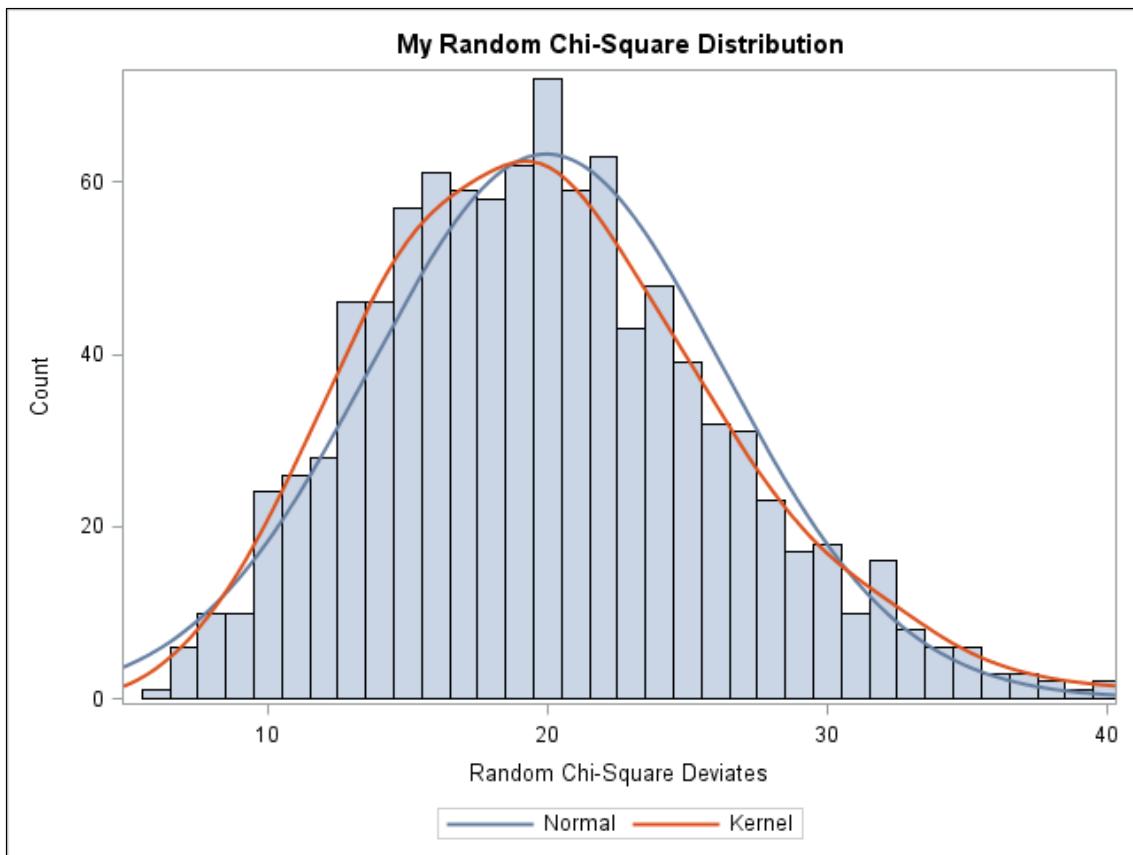
- Create a data table with 1000 random deviates from a chi-square distribution with 20 degrees of freedom and a seed of 123.

```
data sp4r.hist;
call streaminit(123);
do i=1 to 1000;
  rchisq = rand('chisquare',20);
  output;
end;
run;
```

- Use PROC SGPlot to plot a histogram of the data.
 - Alter the appearance of the plot by setting the BINWIDTH= option to 1.
 - Add both a normal and kernel density estimate.
 - Add the title ‘My Random Chi-Square Distribution’.
 - Add the X-axis title ‘Random Chi-Square Deviates’.

- 5) Use X-axis limits of 5 and 40.
- 6) Request the frequency instead of the percent by providing the option SCALE=COUNT in the HISTOGRAM statement.

```
proc sgplot data=sp4r.hist;
  histogram rchisq / binwidth=1 scale=count;
  density rchisq / type=normal;
  density rchisq / type=kernel;
  title 'My Random Chi-Square Distribution';
  xaxis label='Random Chi-Square Deviates' min=5 max=40;
run;
```



4. Plotting Simple Linear Regression Data

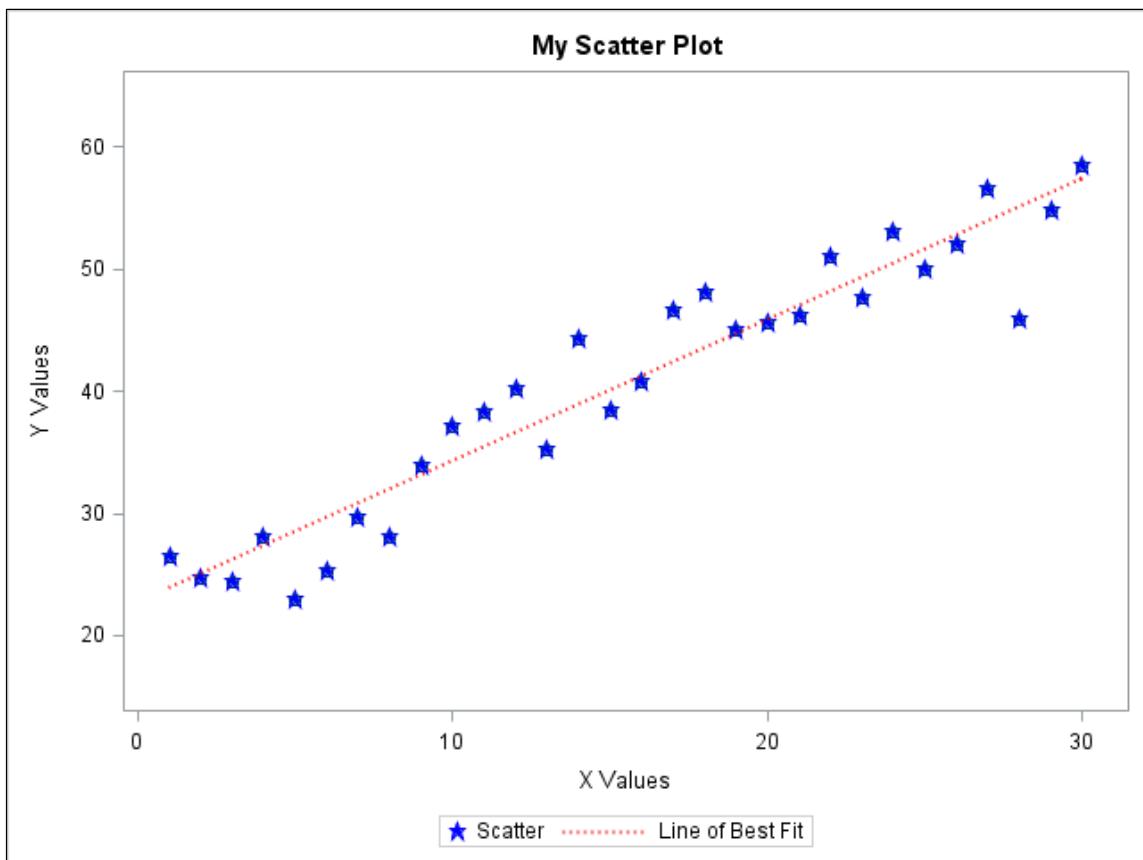
- a. Create a data table with $Y = \beta_0 + \beta_1 X + \varepsilon$ where X ranges from 1 to 30, $\beta_0 = 25$, $\beta_1 = 1$, and $\varepsilon \sim N(\mu = 0, \sigma = 5)$. Keep only the variables X and Y .

```
data sp4r.simple_lin (keep=x y);
  call streaminit(123);
  do x=1 to 30;
    beta01 = 25;
    beta11 = 1;
    y = beta01 + beta11*x + rand('Normal', 0, 5);
    output;
  end;
run;
```

- b. Use PROC SGPlot and the REG statement to plot the line of best fit for the data. Create a plot of the data. Use both the SCATTER and REG statement to plot the points and a line of best fit.
- 1) Enhance the plot by coloring the points blue and using the symbol STARFILLED
 - 2) Color the regression line red and use the pattern DOT.
 - 3) Add a title of your choosing to the X-axis, Y-axis, and the main title.
 - 4) Use the X-axis limits from 0 to 31, and the Y-axis limits from 15 to 65.
 - 5) Name the legend ‘Scatter’ and ‘Line of Best Fit’ for both plot types.

```
proc sgplot data=sp4r.simple_lin;
  scatter x=x y=y / legendlabel='Scatter' name='Scatter'
    markerattrs=(color=blue symbol=starfilled);
  reg x=x y=y / legendlabel='Line of Best Fit' name='Line'
    lineattrs=(color=red pattern=dot);

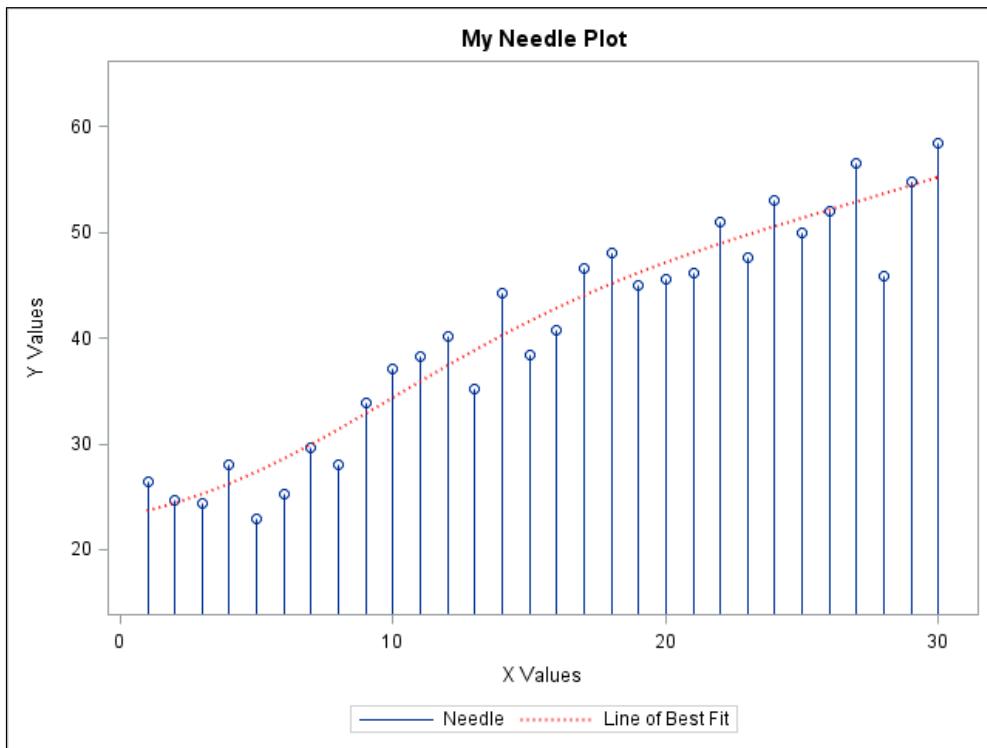
  title 'My Scatter Plot';
  xaxis label='X Values' min=0 max=31;
  yaxis label='Y Values' min=15 max=65;
  keylegend 'Scatter' 'Line';
run;
```



- c. Alter the previous plot by changing the SCATTER statement to NEEDLE and the REG statement to PBSPLINE. (This demonstrates the ease in which plot types can be altered.)

```
proc sgplot data=sp4r.simple_lin;
  needle x=x y=y / legendlabel='Needle' name='Needle'
  markerattrs=(color=blue symbol=starfilled);
  pbspline x=x y=y / legendlabel='Line of Best Fit' name='Line'
  lineattrs=(color=red pattern=dot);

  title 'My Needle Plot';
  xaxis label='X Values' min=0 max=31;
  yaxis label='Y Values' min=15 max=65;
  keylegend 'Needle' 'Line';
run;
```



5. Creating a Bubble Plot

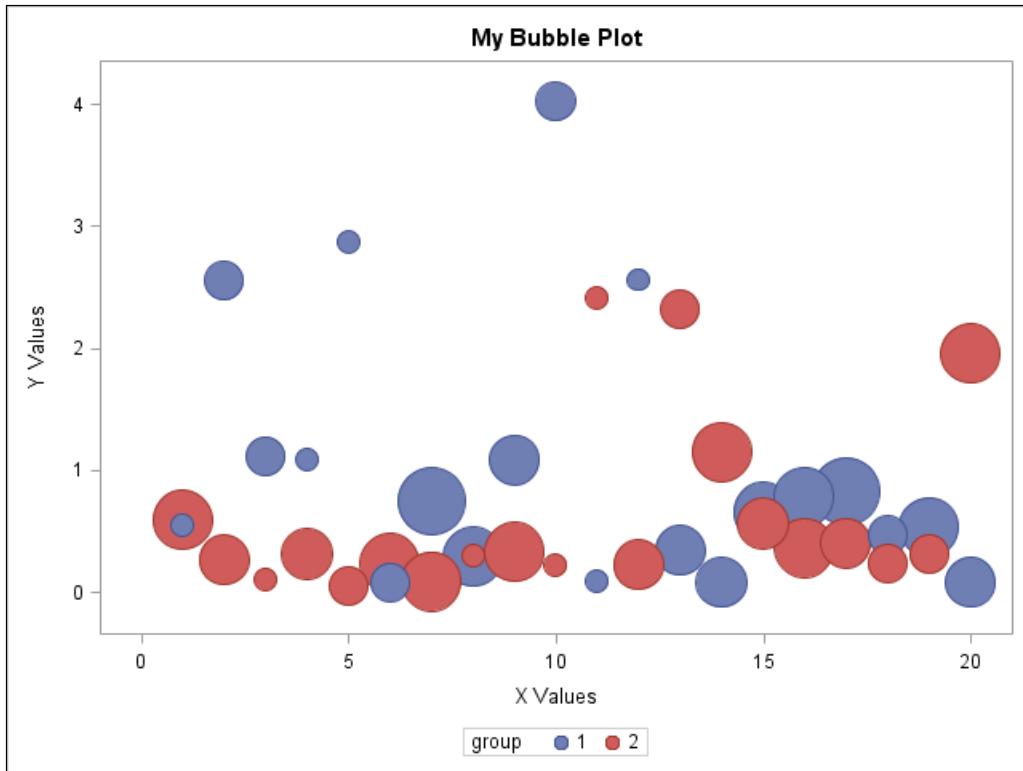
- a. Create a data table with two groups of 20 and the random seed 123. Create two random variables. Let the first be exponential and the second be binomial with parameters 0.5 and 5.

```
data sp4r.bubble;
call streaminit(123);
do group=1 to 2;
  do x=1 to 20;
    y = rand('Exponential');
    z = rand('binomial',.5,5);
    output;
  end;
end;
run;
```

- b. Use the BUBBLE statement to create a bubble plot. Set the SIZE= to the binomial random variable. Also, specify the GROUP= option based on the two separate groups. Finally, provide the plot with titles for the X-axis, Y-axis, and main title.

```
proc sgplot data=sp4r.bubble;
  bubble x=x y=y size=z / group=group;

  title 'My Bubble Plot';
  xaxis label='X Values';
  yaxis label='Y Values';
run;
```



6. Using PROC SGSCATTER

- a. Create a data table with 300 observations and a seed of 123.
- 1) Let **X** be the deviates from the standard normal distribution.
 - 2) Produce a variable **Y1**, which is **X** plus standard normal deviates.
 - 3) Produce another variable such that **Y2** is $5*X$ plus standard normal deviates.

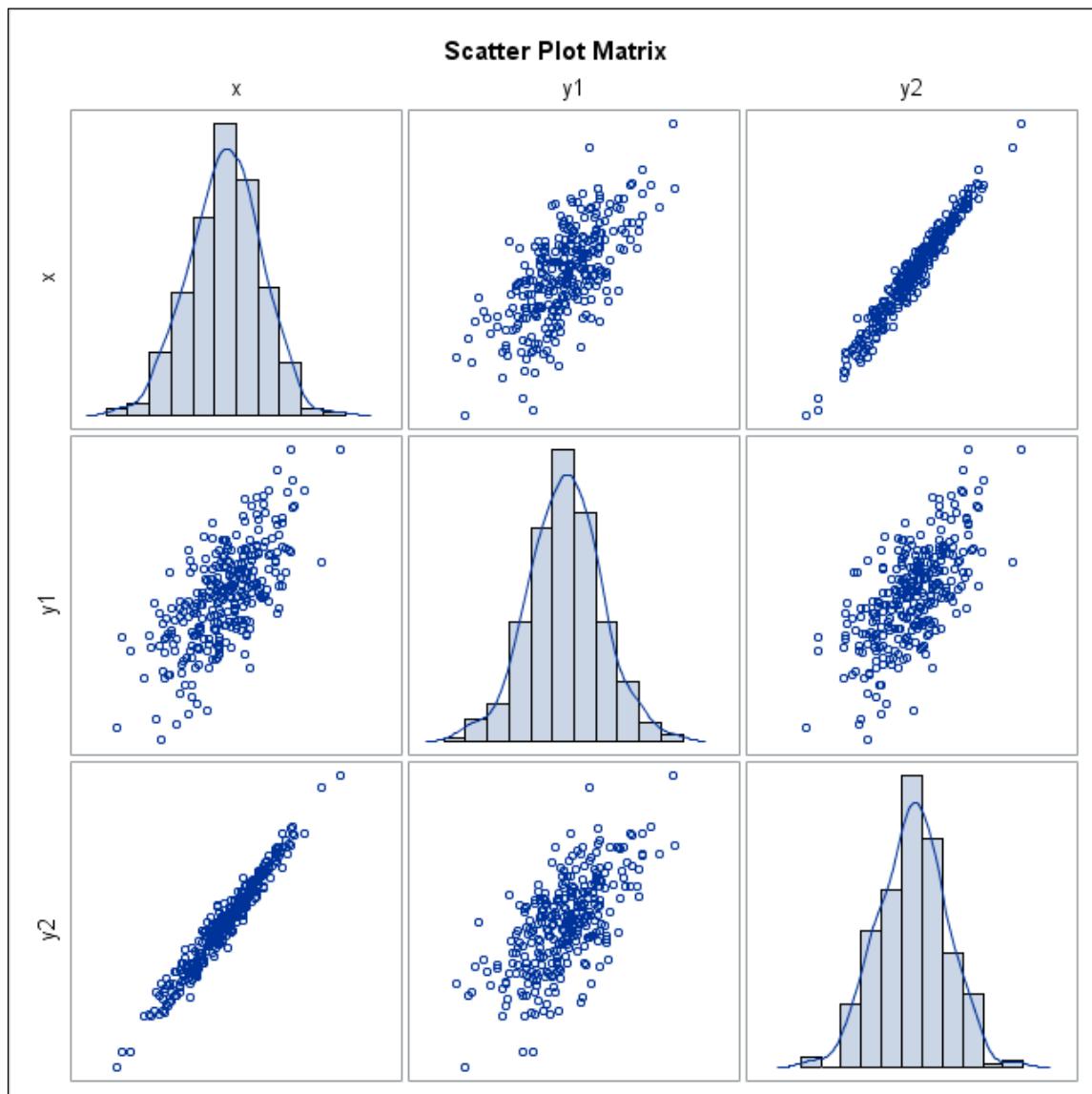
```
data sp4r.random;
  call streaminit(123);
  do i=1 to 300;
    x = rand('Normal');
    y1 = x + rand('Normal');
    y2 = 5*x + rand('Normal');
    output;
  end;
run;
```

- b. Use PROC SGSCATTER to create a scatter plot matrix of **X**, **Y1**, and **Y2**. Include histograms and kernel density estimates on the diagonal. (Hint: Look up the **DIAGONAL=** option in the **MATRIX** statement of the SGSCATTER procedure.)

```
proc sgscatter data=sp4r.random;
  matrix x y1 y2 / diagonal=(histogram kernel);
  title 'Scatter Plot Matrix';
run;
title;
```

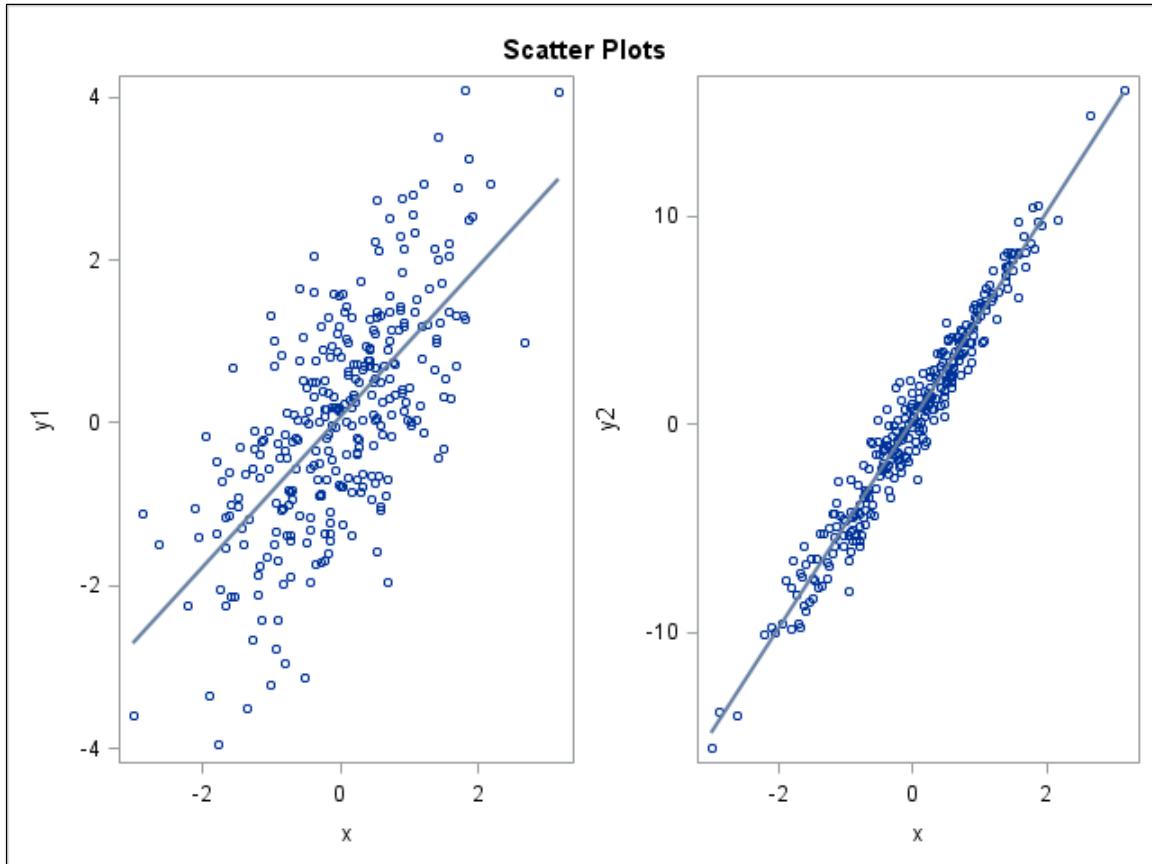
Selected PROC SGSCATTER statement and option:

MATRIX specifies the variables used to create a scatter plot matrix. Use the **DIAGONAL=** option to include a histogram, density estimates, or both as the diagonal elements of the scatter plot matrix.



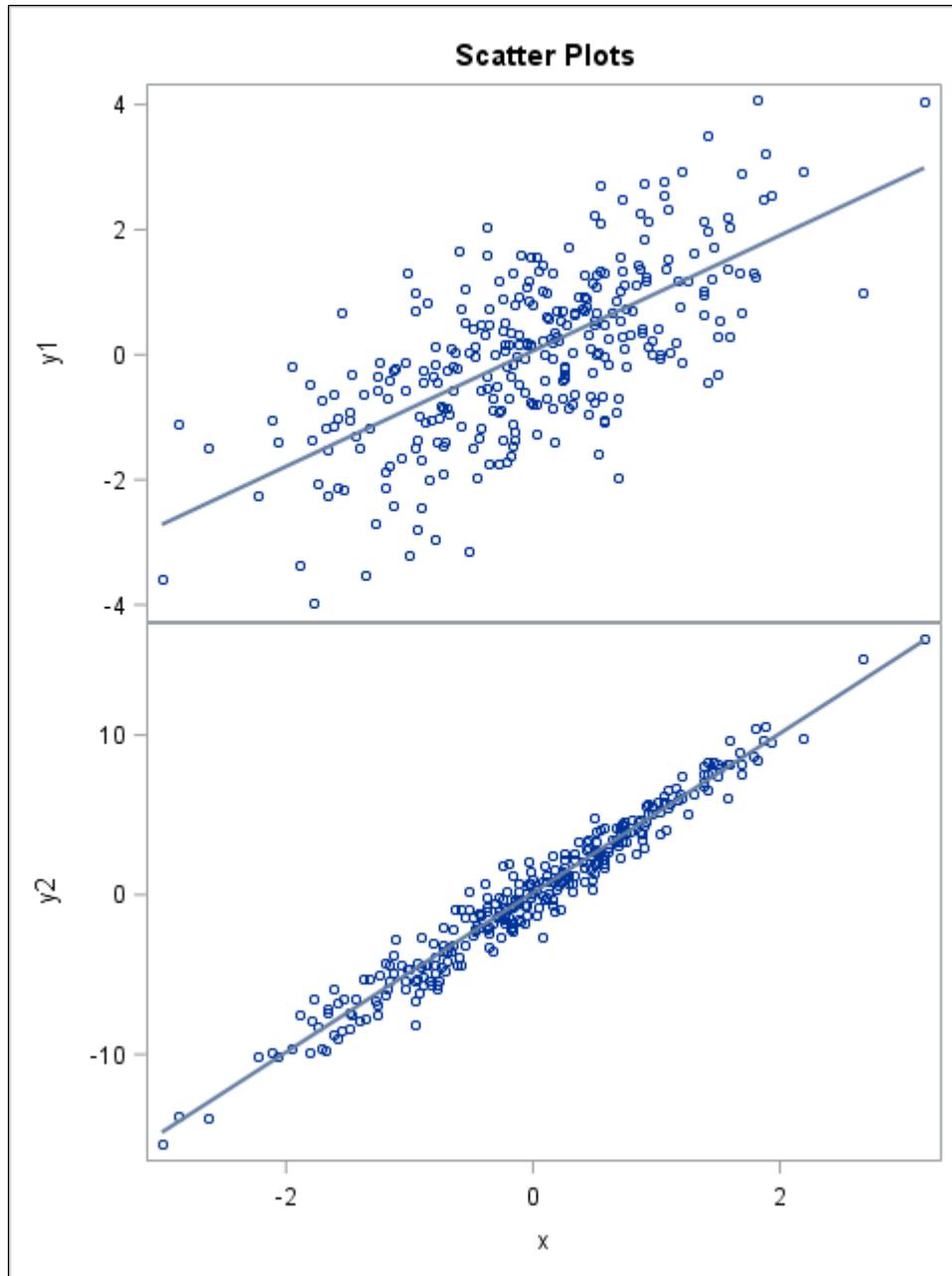
- c. Use PROC SGSCATTER to create side-by-side scatter plots of **Y1** by **X** and **Y2** by **X** with the PLOT statement. Add the regression line to both plots with the REG option.

```
proc sgscatter data=sp4r.random;
  plot (y1 y2) * x / reg;
  title 'Scatter Plots';
run;
title;
```



- d. Use PROC SGSCATTER and the COMPARE statement to create the same scatter plot with shared axes.

```
proc sgscatter;
  compare y=(y1 y2) x=x / reg;
  title 'Scatter Plots';
run;
title;
```



7. Using PROC SGPANEL

- Instead of creating **Y1** and **Y2** as separate variables from the previous exercise, stack the variables in a single column denoted **Y** using a nested DO loop.
 - Create a categorical variable called **Year** with groups 1 and 2.
 - Generate 300 observations for each group with a random seed of 123.
 - Let **X** be the deviates from a standard normal distribution.
 - Use IF-THEN/ELSE syntax to let **Y** be **X** plus standard normal deviates if **Year** is 1 and let **Y** be $5 \times \mathbf{X}$ plus standard normal deviates otherwise.

```

data sp4r.random;
call streaminit(123);
do year=1 to 2;
  do j=1 to 300;
    x = rand('Normal');
    if year=1 then y = x + rand('Normal');
    if year=2 then y = 5*x + rand('Normal');
    output;
  end;
end;
run;

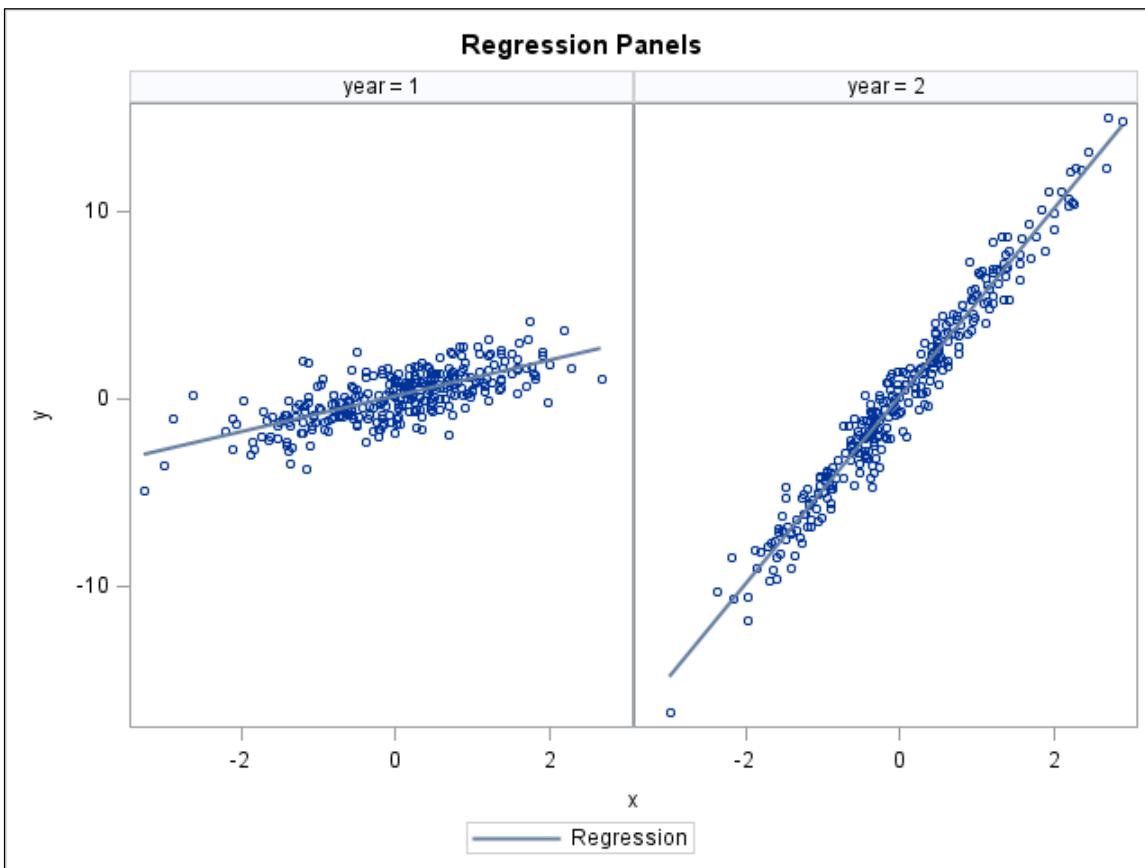
```

- b. Use PROC SGPANEL to create a regression panel by year.

```

proc sgpanel data=sp4r.random;
  panelby year;
  reg x=x y=y;
  title 'Regression Panels';
run;
title;

```



End of Solutions

Solutions to Student Activities (Polls/Quizzes)

4.01 Poll – Correct Answer

Does the following DO loop create a data table with a sequence from 50 to 100 by 5 with the variable name **myloop**?

- Yes
- No

```
data doloop;
  do myloop=50 to 100 by 5;
  end;
run;
```

15

The new data table has one row and **myloop** is 105.

4.02 Multiple Choice Poll – Correct Answer

What is the dimension of the data set created below?

- a. 12x2
- b. 6x3
- c. 8x3
- d. 12x3

```
data random;
  do i=1 to 3;
    do j=1 to 2;
      do k=1 to 2;
        output;
      end;
    end;
  end;
run;
```

17

4.03 Poll – Correct Answer

Will the SAS functions `rand('Beta',5,7)` and `CDF('Beta',.3,5,7)` reproduce the R functions `rbeta(1,5,7)` and `pbeta(.3,5,7)`?

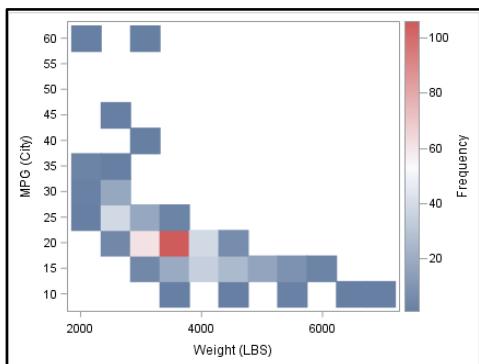
- Yes
- No

25

4.04 Multiple Choice Poll – Correct Answer

Navigate to the SGPlot procedure Help documentation and examine the plotting statements. Which statement is used to create the following plot?

- a. HEATMAP
- b. POLYGON
- c. BUBBLE
- d. BLOCK

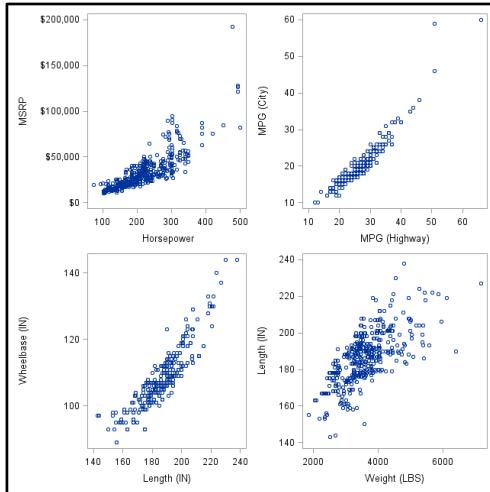


40

4.05 Multiple Choice Poll – Correct Answer

Which SGPlot procedure statement is used to create the plot?

- a. MATRIX
- b. PLOT**
- c. COMPARE



60

Chapter 5 Descriptive Procedures, Output Delivery System, and Macros

5.1 CORR, FREQ, MEANS, and UNIVARIATE Procedures.....	5-3
Demonstration: Ames Home Sales Data Exploration	5-15
5.2 Output Delivery System (ODS)	5-20
Demonstration: Selecting Output with ODS TRACE	5-24
Demonstration: ODS Tables and Graphics	5-26
Demonstration: Creating Data Tables with ODS.....	5-30
5.3 Creating Macro Variables	5-32
Demonstration: Creating and Using Macro Variables	5-42
5.4 Creating Macro Programs	5-45
Demonstration: A Macro Program to Generate Data, Summary Statistics, and Plots.....	5-55
Demonstration: A Macro Program for Iterative Processing.....	5-58
Exercises	5-60
5.5 Solutions	5-67
Solutions to Exercises	5-67
Solutions to Student Activities (Polls/Quizzes)	5-81

5.1 CORR, FREQ, MEANS, and UNIVARIATE Procedures

Objectives

- Use PROC CORR to compute correlation and covariance matrices.
- Use PROC FREQ to analyze categorical data.
- Use PROC MEANS to compute summary statistics.
- Use PROC UNIVARIATE to explore continuous data.

3

Motivation

Use a PROC step to explore the data.

PROC Step



4

Duplicating the R Script

```
#Compute correlation and covariance matrices
cor(cbind(cars$Horsepower,cars$weight,cars$Length))
cov(cbind(cars$Horsepower,cars$weight,cars$Length))

#Create frequency tables
table(cars$origin)
table(cars$type)
#Create contingency tables
table(cars$origin,cars$type)

#Create QQ-plot
qqnorm(cars$Hits)

#Compute summary statistics
mean(cars$mpg_city)
median(cars$mpg_city)
mode(cars$mpg_city)
range(cars$mpg_city)
var(cars$mpg_city)
sd(cars$mpg_city)
sum(cars$mpg_city)
min(cars$mpg_city)
max(cars$mpg_city)
quantile(cars$mpg_city,c(.01,.05,.1,.25,.5,.75,.05,.9,.99))
```

5

CORR Procedure

Use PROC CORR to calculate the correlation matrix.

```
cor(cbind(cars$Horsepower,cars$weight,cars$Length))

proc corr data=sp4r.cars;
  var horsepower weight length;
run;
```

PROC CORR DATA=SAS-data-set <options>;
VAR variable-1 ... variable-n;
RUN;

6

Selected SAS statement:

VAR specifies numeric variables for which to compute correlation coefficients.

 If no VAR statement is included, all numeric variables in the data set are included.

CORR Procedure

View the default PROC CORR output.

```
Source on Save | Run | Source
cor(cbind(cars$Horsepower,cars$Weight,cars$Length))
```

```
proc corr data=sp4r.cars;
  var horsepower weight length;
run;
```

PROC CORR Output

Simple Statistics						
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
Horsepower	428	215.88551	71.83603	92399	73.00000	500.00000
Weight	428	3578	758.98321	1531364	1850	7190
Length	428	186.36215	14.35799	79763	143.00000	238.00000

Pearson Correlation Coefficients, N = 428 Prob > r under H0: Rho=0			
	Horsepower	Weight	Length
Horsepower	1.00000	0.63080	0.38155 <.0001
Weight	0.63080 <.0001	1.00000	0.69002 <.0001
Length	0.38155 <.0001	0.69002	1.00000

7

The default CORR procedure provides a table of simple statistics and a correlation matrix. The correlation matrix includes the *p*-value for the hypothesis test $H_0: \text{Rho}=0$ for each correlation value.

CORR Procedure

Use the COV option to print the covariance matrix.

```
Source on Save | Run | Source
cor(cbind(cars$Horsepower,cars$Weight,cars$Length))
cov(cbind(cars$Horsepower,cars$Weight,cars$Length))
```

```
proc corr data=sp4r.cars cov;
  var horsepower weight length;
run;
```

Partial PROC CORR Output

Covariance Matrix, DF = 427			
	Horsepower	Weight	Length
Horsepower	5160.4154	34392.4654	393.5427
Weight	34392.4654	576055.5201	7519.4830
Length	393.5427	7519.4830	206.1519

8

FREQ Procedure

Use PROC FREQ to produce frequency tables.

```
Source on Save | Run | Source
table(cars$origin)
table(cars$type)
```

```
proc freq data=sp4r.cars;
  tables origin type;
run;
```

```
PROC FREQ DATA=SAS-data-set <options>;
  TABLES variable-1 ... variable-n / <options>;
RUN;
```

9

Selected PROC FREQ statement:

TABLES The FREQ procedure produces a one-way frequency table for each variable (numeric or categorical) named in the statement. Multiple variables can be specified.

 If the TABLES statement is omitted, a one-way frequency table is produced for every variable in the data set. This is seldom preferred.

FREQ Procedure

Viewing the default PROC FREQ output.

```
Source on Save | Run | Source
table(cars$origin)
table(cars$type)
```

```
proc freq data=sp4r.cars;
  tables origin type;
run;
```

Origin	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Asia	158	36.92	158	36.92
Europe	123	28.74	281	65.65
USA	147	34.35	428	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	0.70	3	0.70
SUV	60	14.02	63	14.72
Sedan	262	61.21	325	75.93
Sports	49	11.45	374	87.38
Truck	24	5.61	398	92.99
Wagon	30	7.01	428	100.00

10

FREQ Procedure

Use options in the TABLES statement to suppress the display of selected default statistics.

```
TABLES variable(s) / options ;
```

Option	Description
NOCUM	Suppresses the cumulative statistics.
NOPERCENT	Suppresses the percentage display.

11

FREQ Procedure

Use PROC FREQ to produce cross tabulation tables.

```
proc freq data=sp4r.cars;
  tables origin*type;
run;
```

```
PROC FREQ DATA=SAS-data-set <options>;
  TABLES variable-1* ... *variable-n / <options>;
  RUN;
```

12

Create a cross tabulation table for two variables by using the * symbol.

FREQ Procedure

View the default PROC FREQ output.

```
proc freq data=sp4r.cars;
  tables origin*type;
run;
```

13

Origin	Type							Total
	Hybrid	SUV	Sedan	Sports	Truck	Wagon		
	Frequency	Percent	Row Pct	Col Pct				
	3	25	94	17	8	11	158	
Asia	0.70	5.84	21.96	3.97	1.87	2.57	36.92	
	1.90	15.82	59.49	10.76	5.06	6.96		
	100.00	41.67	35.88	34.69	33.33	36.67		
Europe	0	10	78	23	0	12	123	
	0.00	2.34	18.22	5.37	0.00	2.80	28.74	
	0.00	8.13	63.41	18.70	0.00	9.76		
	0.00	16.67	29.77	46.94	0.00	40.00		
USA	0	25	90	9	16	7	147	
	0.00	5.84	21.03	2.10	3.74	1.64	34.35	
	0.00	17.01	61.22	6.12	10.88	4.76		
	0.00	41.67	34.35	18.37	66.67	23.33		
Total	3	60	262	49	24	30	428	
	0.70	14.02	61.21	11.45	5.61	7.01	100.00	

The default contingency table includes the frequency for each cell, the percent, row percent, and column percent. In addition, the total frequency and percent is displayed for each row and column outside the table.

Options to Suppress Statistics

Use options in the TABLES statement to suppress the display of selected default statistics.

```
TABLES variable(s) / options ;
```

Option	Description
NOROW	Suppresses the display of the row percentage.
NOCOL	Suppresses the display of the column percentage.
NOPERCENT	Suppresses the percentage display.
NOFREQ	Suppresses the frequency display.

14

FREQ Procedure

Use options in the TABLES statement to suppress output.

```
proc freq data=sp4r.cars;
  tables origin*type / norow nocol nopercnt;
run;
```

PROC FREQ Output

Frequency		Table of Origin by Type						
Origin		Type						Total
		Hybrid	SUV	Sedan	Sports	Truck	Wagon	
Asia		3	25	94	17	8	11	158
Europe		0	10	78	23	0	12	123
USA		0	25	90	9	16	7	147
Total		3	60	262	49	24	30	428

15

Use of the NOROW, NOCOL, and NOPERCENT options provides a table that is identical to the table() function in R.

FREQ Procedure

Use the NLEVELS option to print the number of levels for each variable.

```
Source on Save | Run | Source
length(levels(cars$origin))
length(levels(cars$type))
```

```
proc freq data=sp4r.cars nlevels;
  tables origin*type /
    norow nocol nopercnt nowrap;
run;
```

PROC FREQ Output

Number of Variable Levels	
Variable	Levels
Origin	3
Type	6

16

Use the NOPRINT option in the TABLES statement to suppress the tables output when you use the NLEVELS option.

5.01 Quiz

Navigate to the TABLES statement on the SAS documentation page for the FREQ procedure.
What does the PROC step below do?

```
proc freq data=sp4r.cars;
  tables origin*type / chisq;
run;
```

17

For a list of the TABLES statement options, go to
http://support.sas.com/documentation/cdl/en/procstat/66703/HTML/default/viewer.htm#procstat_freq_syntax08.htm.

Also, the default tests and output are described here:

http://support.sas.com/documentation/cdl/en/procstat/66703/HTML/default/viewer.htm#procstat_freq_details08.htm

MEANS Procedure

Use PROC MEANS to reproduce the R functions.

Mean()	Median()	Mode()	Range()	Var()
sd()	Sum()	Min()	Max()	Quantile()

```
proc means data=sp4r.cars;
  var mpg_city mpg_highway;
run;
```

```
PROC MEANS DATA=SAS-data-set <options>;
  VAR variable-1 ... variable-n;
RUN;
```

19

MEANS Procedure

View the PROC MEANS default output.

Mean()	Median()	Mode()	Range()	Var()
sd()	Sum()	Min()	Max()	Quantile()

```
proc means data=sp4r.cars maxdec=2;
  var mpg_city mpg_highway;
run;
```

PROC MEANS Output

Variable	N	Mean	Std Dev	Minimum	Maximum
MPG_City	428	20.06	5.24	10.00	60.00
MPG_Highway	428	26.84	5.74	12.00	66.00

20

The default output displays the number of observations with nonmissing values, the mean, standard deviation, minimum, and maximum.

Selected PROC MEANS statements:

VAR specifies the numeric variables for which to compute summary statistics.

MAXDEC specifies the number of decimals to display for each statistic.

MEANS Procedure

Use the following PROC MEANS statement options to request the desired statistics:

Descriptive Statistic Keywords				
CLM	CSS	CV	LCLM	MAX
MEAN	MIN	MODE	N	NMISS
KURTOSIS	RANGE	SKEWNESS	STDDEV	STDERR
SUM	SUMWGT	UCLM	USS	VAR

Quantile Statistic Keywords				
MEDIAN P50	P1	P5	P10	Q1 P25
Q3 P75	P90	P95	P99	QRANGE

21

Use the ALPHA= option when you use confidence limit options.

MEANS Procedure

Use options in the PROC MEANS statement to request specific statistics only.

```
proc means data=sp4r.cars maxdec=2
            mean median var;
    var mpg_city mpg_highway;
run;
```

PROC MEANS Output

Variable	Mean	Median	Variance
MPG_City	20.06	19.00	27.44
MPG_Highway	26.84	26.00	32.96

22

Customize the PROC MEANS results by specifying desired statistics in the PROC MEANS statement.

5.02 Multiple Choice Poll

Which SAS procedures are used to reproduce the R functions min(), cov(), table(), and sd()?

- a. FREQ, MEANS, CORR, MEANS
- b. MEANS, CORR, FREQ, MEANS
- c. MEANS FREQ, CORR, MEANS
- d. CORR, FREQ, FREQ, MEANS

23

UNIVARIATE Procedure

The default PROC UNIVARIATE output displays

- moments
- basic statistical measures
- tests for location
- quantiles
- extreme observations.

```
proc univariate data=sp4r.cars;
  var mpg_city mpg_highway;
run;
```

PROC UNIVARIATE DATA=SAS-data-set <options>;
 VAR variable-1 ... variable-n;
RUN;

25

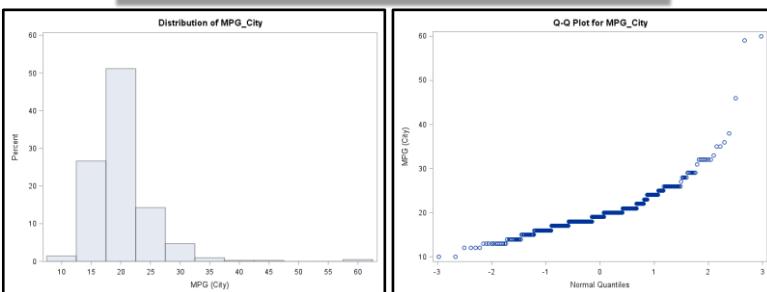
The default output is discussed in the subsequent demonstration.

UNIVARIATE Procedure

Use the HISTOGRAM and QQPLOT statements to request graphical output.

```
Source on Save | Run | Source
hist(cars$mpg_city)
qqnorm(cars$mpg_city)
```

HISTOGRAM variable-1 ... variable-n / <options>;
QQPLOT variable-1 ... variable-n / <options>;



26

The Ames Home Sales Data Set



27

The data set **AmesHousing** is a random sample of 300 houses sold in Ames, Iowa from 2006 to 2010. The original data set was collected by Dr. Dean Decock of Truman State University in Kirksville, Missouri, USA. The full description of the data set is provided in the *Journal of Statistics Education*. The original data set contains 2,930 observations with information about the sale of residential properties.



Ames Home Sales Data Exploration

SP4R05d01.sas

Use the **AmesHousing** data set to practice using the CORR, FREQ, MEANS, and UNIVARIATE procedures.

1. Use PROC CONTENTS to list all the variables in the **AmesHousing** data set.

```
proc contents data=sp4r.ameshousing varnum;
run;
```

Variables in Creation Order

#	Variable	Type	Len	Format	Informat
1	PID	Char	10	\$CHAR10.	\$CHAR10.
2	Lot_Area	Num	8	BEST12.	BEST12.
3	House_Style	Char	6	\$CHAR6.	\$CHAR6.
4	Overall_Qual	Num	8	BEST12.	BEST12.
5	Overall_Cond	Num	8	BEST12.	BEST12.
6	Year_Built	Num	8	BEST12.	BEST12.
7	Heating_QC	Char	2	\$CHAR2.	\$CHAR2.
8	Central_Air	Char	1	\$CHAR1.	\$CHAR1.
9	Gr_Liv_Area	Num	8	BEST12.	BEST12.
10	Bedroom_AbvGr	Num	8	BEST12.	BEST12.
11	Fireplaces	Num	8	BEST12.	BEST12.
12	Garage_Area	Num	8	BEST12.	BEST12.
13	Mo_Sold	Num	8	BEST12.	BEST12.
14	Yr_Sold	Num	8	BEST12.	BEST12.
15	SalePrice	Num	8	BEST12.	BEST12.
16	Basement_Area	Num	8	BEST12.	BEST12.
17	Full_Bathroom	Num	8	BEST12.	BEST12.
18	Half_Bathroom	Num	8	BEST12.	BEST12.
19	Total_Bathroom	Num	8	BEST12.	BEST12.
20	Deck_Porch_Area	Num	8	BEST12.	BEST12.
21	Age_Sold	Num	8	BEST12.	BEST12.
22	Season_Sold	Num	8	BEST12.	BEST12.
23	Garage_Type_2	Char	8	\$CHAR8.	\$CHAR8.
24	Foundation_2	Char	16	\$CHAR16.	\$CHAR16.
25	Masonry_Veneer	Char	1	\$CHAR1.	\$CHAR1.
26	Lot_Shape_2	Char	10	\$CHAR10.	\$CHAR10.
27	House_Style2	Char	6	\$CHAR6.	\$CHAR6.
28	Overall_Qual2	Num	8	BEST12.	BEST12.
29	Overall_Cond2	Num	8	BEST12.	BEST12.
30	Log_Price	Num	8	BEST12.	BEST12.
31	Bonus	Num	8	BEST12.	BEST12.
32	score	Num	8		

The variables in the data set are listed below:

PID

Lot_Area Lot size in square feet

House_Style Style of dwelling

Overall_Qual	Overall material and finish of the house
Overall_Cond	Overall condition of the house
Year_Built	Original construction year
Heating_QC	Heating quality and condition
Central_Air	Presence of central air conditioning
Gr_Liv_Area	Above ground living area in square feet
Bedroom_AbvGr	Bedrooms above ground
Fireplaces	Number of fireplaces
Garage_Area	Size of garage in square feet
Mo_Sold	Month sold (MM)
Yr_Sold	Year sold (YYYY)
SalePrice	Sale price in dollars
Basement_area	Basement area in square feet
Full_Bathroom	Number of full bathrooms
Half_Bathroom	Number of half bathrooms
Total_Bathroom	Total number of bathrooms
Deck_Porch_Area	Total area of decks and porches in square feet
Age_Sold	Age of house when sold, in years
Season_Sold	Season when the house sold
Garage_Type_2	Garage attached, detached, or NA
Foundation_2	Foundation type
Masonry_Veneer	Masonry veneer or not
Lot_Shape_2	Regular or irregular lot shape
House_Style2	Style of dwelling
Overall_Qual2	Overall material and finish of the house
Overall_Cond2	Overall condition of the house
Log_Price	Natural log of the sale price
Bonus	Sale price > \$175,000

Score

2. Use PROC UNIVARIATE to explore the **SalePrice** variable further. Use the HISTOGRAM, INSET, and QQPLOT statements to print pertinent graphics.

```
proc univariate data=sp4r.ameshousing;
  var saleprice;
  histogram saleprice / normal kernel;
  inset n mean std / position=ne;
  qqplot saleprice / normal(mu=est sigma=est);
run;
```

Selected PROC UNIVARIATE statements and options:

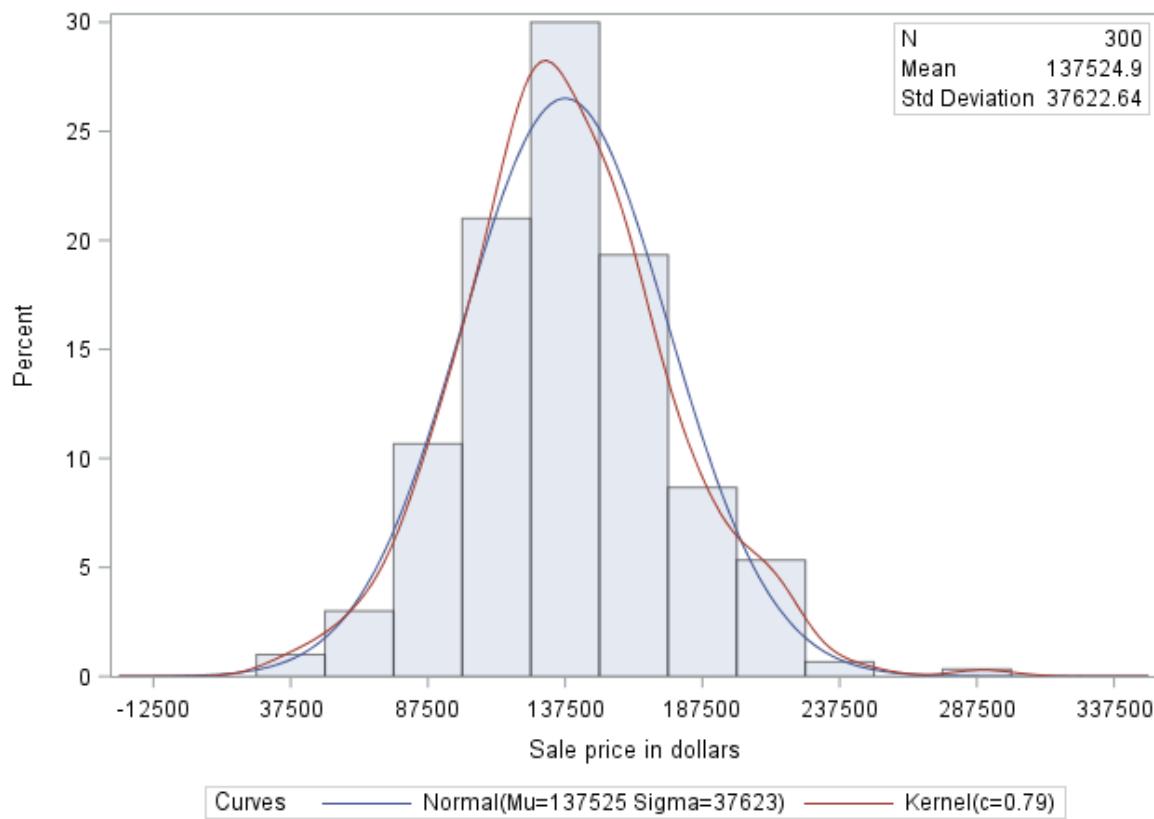
- VAR specifies numeric variables to analyze.
- HISTOGRAM specifies a numeric variable to create a histogram. Use the NORMAL and KERNEL option to overlay a normal density and kernel density estimate.
- INSET specifies which statistics to include in the histogram plot. Use the POSITION= option to provide a location. Provide the option with a compass direction (NE = North East).
- QQPLOT specifies numeric variables to create a QQ-plot. Use the NORMAL option to add a line to the QQ-plot. Use the MU= and SIGMA= options to specify the parameters of the distribution for which quantiles are compared.

 The QQPLOT statement option can specify different distributions for quantile comparison.

Variable: SalePrice (Sale price in dollars)			
Moments			
N	300	Sum Weights	300
Mean	137524.867	Sum Observations	41257460
Std Deviation	37622.6431	Variance	1415463276
Skewness	0.29726388	Kurtosis	0.72287774
Uncorrected SS	6.09715E12	Corrected SS	4.23224E11
Coeff Variation	27.3569748	Std Error Mean	2172.14431
Basic Statistical Measures			
Location		Variability	
Mean	137524.9	Std Deviation	37623
Median	135000.0	Variance	1415463276
Mode	110000.0	Range	255000
		Interquartile Range	45475
Note: The mode displayed is the smallest of 2 modes with a count of 6.			
Tests for Location: Mu0=0			
Test	-Statistic-	-----p Value-----	
Student's t	t 63.31295	Pr > t	<.0001
Sign	M 150	Pr >= M	<.0001
Signed Rank	S 22575	Pr >= S	<.0001
Quantiles (Definition 5)			
Level	Quantile		
100% Max	290000		
99%	227500		
95%	207000		
90%	187300		
75% Q3	159475		
50% Median	135000		
25% Q1	114000		
10%	91150		
5%	80000		
1%	48500		
0% Min	35000		
Extreme Observations			

-----Lowest----- -----Highest-----

Value	Obs	Value	Obs
35000	294	218000	184
39300	190	220000	106
45000	77	235000	151
52000	130	245000	54
59000	70	290000	123

Distribution of SalePrice**Parameters for Normal Distribution**

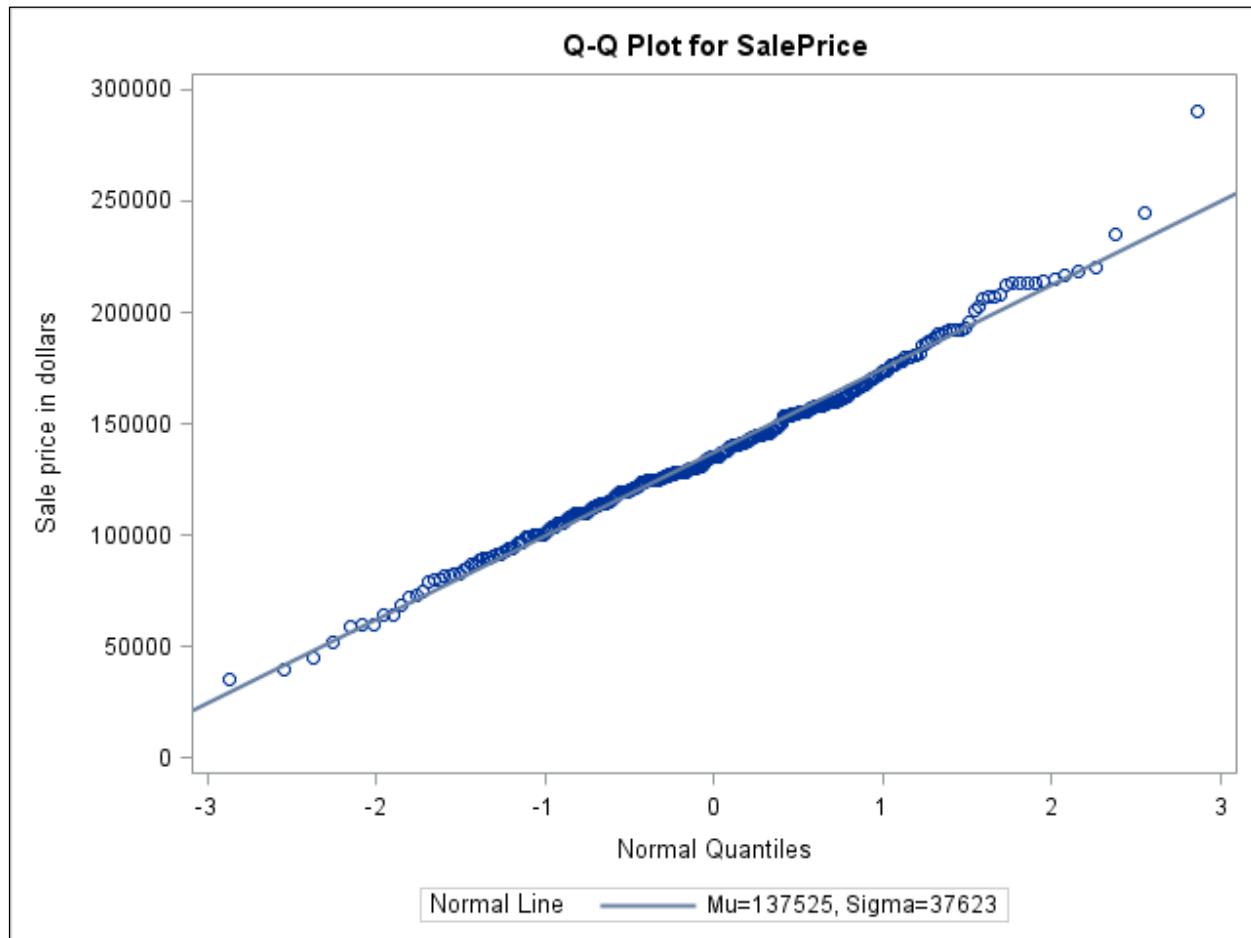
Parameter	Symbol	Estimate
Mean	Mu	137524.9
Std Dev	Sigma	37622.64

Goodness-of-Fit Tests for Normal Distribution

Test	----Statistic----		----p Value----	
	D	0.04433755	Pr > D	>0.150
Kolmogorov-Smirnov				
Cramer-von Mises	W-Sq	0.11943883	Pr > W-Sq	0.064
Anderson-Darling	A-Sq	0.69161031	Pr > A-Sq	0.074

Quantiles for Normal Distribution

Percent	Quantile	
	Observed	Estimated
1.0	48500.0	50001.5
5.0	80000.0	75641.1
10.0	91150.0	89309.5
25.0	114000.0	112148.8
50.0	135000.0	137524.9
75.0	159475.0	162901.0
90.0	187300.0	185740.2
95.0	207000.0	199408.6
99.0	227500.0	225048.2



5.2 Output Delivery System (ODS)

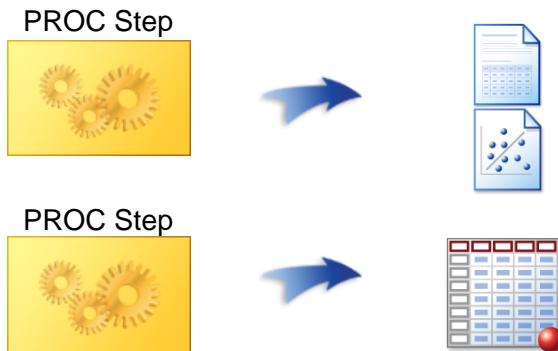
Objectives

- Use ODS SELECT to request specific output.
- Use ODS OUTPUT to create a new SAS data table.
- Use the OUTPUT statement to customize the creation of a new SAS data table.

30

Motivation

- Customize PROC step output.
- Create new data tables from the PROC step results.



31

Duplicating the R Script

```

Source | Search | Edit | File | Help | Source
#Create object
mylm = lm(y~x)

#Print default output
mylm

#What else is in the object?
names(mylm)

#Pull desired output from the object
mylm$residuals

#Create data frame from the output
mylm_res = data.frame(mylm$residuals)

```

32

Output Delivery System (ODS)

Procedures and DATA steps simply produce raw data.
ODS supplies the structure and format for the output.

Partial PROC CORR Output

Pearson Correlation Coefficients, N = 428			
	Horsepower	Weight	Length
Horsepower	1.00000	0.63080 <.0001	0.38155 <.0001
Weight	0.63080 <.0001	1.00000	0.69002 <.0001
Length	0.38155 <.0001	0.69002	1.00000

33

The CORR procedure did not create the tables, formatting, or even the labels. The CORR procedure simply produced the raw data for the table cells.

Output Delivery System (ODS)

Use the Output Delivery System to perform the following tasks:

- alter the appearance of the output
 - style, color, font, and so on
- change the destination of the output
- select output
- create new data tables from PROC step output

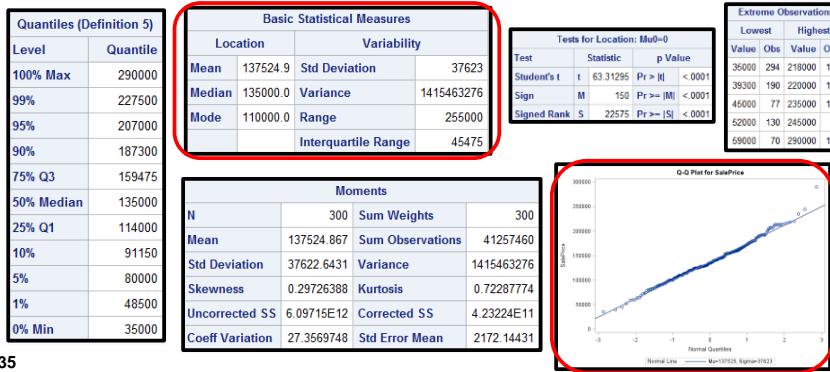
34

Focus on the two latter categories.

Output Delivery System (ODS)

Use the ODS SELECT statement to choose output before running the PROC step.

ODS SELECT object-name-1 ... object-name-n;



35

Choose which tables and graphs to generate by providing the object name in the ODS SELECT statement. Perhaps you want to create only the Basic Statistical Measures table and the Q-Q-plot.



This is identical to using the \$ symbol in R to pull output from an object.

Output Delivery System (ODS)

Print the names of the tables and graphs to the SAS log using the ODS TRACE statement.

```
ods trace on; ODS TRACE ON;  
  
proc univariate data=ameshousing;  
  var saleprice;  
  qqplot saleprice /normal(mu=est sigma=est);  
run;  
  
ods trace off; ODS TRACE OFF;
```

36

You can keep the ODS TRACE ON statement active throughout your SAS session. It does not need to be turned off. The log displays the table and graph names of all subsequent output for any PROC step.



Selecting Output with ODS TRACE

SP4R05d02.sas

1. Use ODS TRACE to view the table and plot names of PROC UNIVARIATE. Use a QQPLOT statement and analyze only the **SalePrice** variable.

```
ods trace on;
proc univariate data=sp4r.ameshousing;
  var saleprice;
  qqplot saleprice / normal(mu=est sigma=est);
run;
ods trace off;
```

```
Output Added
-----
Name:      Moments
Label:     Moments
Template:  base.univariate.Moments
Path:      Univariate.SalePrice.Moments
-----

Output Added:
-----
Name:      BasicMeasures
Label:    Basic Measures of Location and Variability
Template: base.univariate.Measures
Path:      Univariate.SalePrice.BasicMeasures
-----

Output Added:
-----
Name:      TestsForLocation
Label:    Tests For Location
Template: base.univariate.Location
Path:      Univariate.SalePrice.TestsForLocation
-----

Output Added:
-----
Name:      Quantiles
Label:     Quantiles
Template: base.univariate.Quantiles
Path:      Univariate.SalePrice.Quantiles
-----

Output Added:
-----
Name:      ExtremeObs
Label:    Extreme Observations
Template: base.univariate.ExtObs
Path:      Univariate.SalePrice.ExtremeObs
-----
```

```
Output Added:
```

```
Name:      QQPlot
Label:    Panel 1
Template: base.univariate.Graphics.QQPlot
Path:     Univariate.SalePrice.QQPlot.QQPlot
```

2. Use ODS SELECT and PROC UNIVARIATE to print only the BasicMeasures table and the QQPlot.

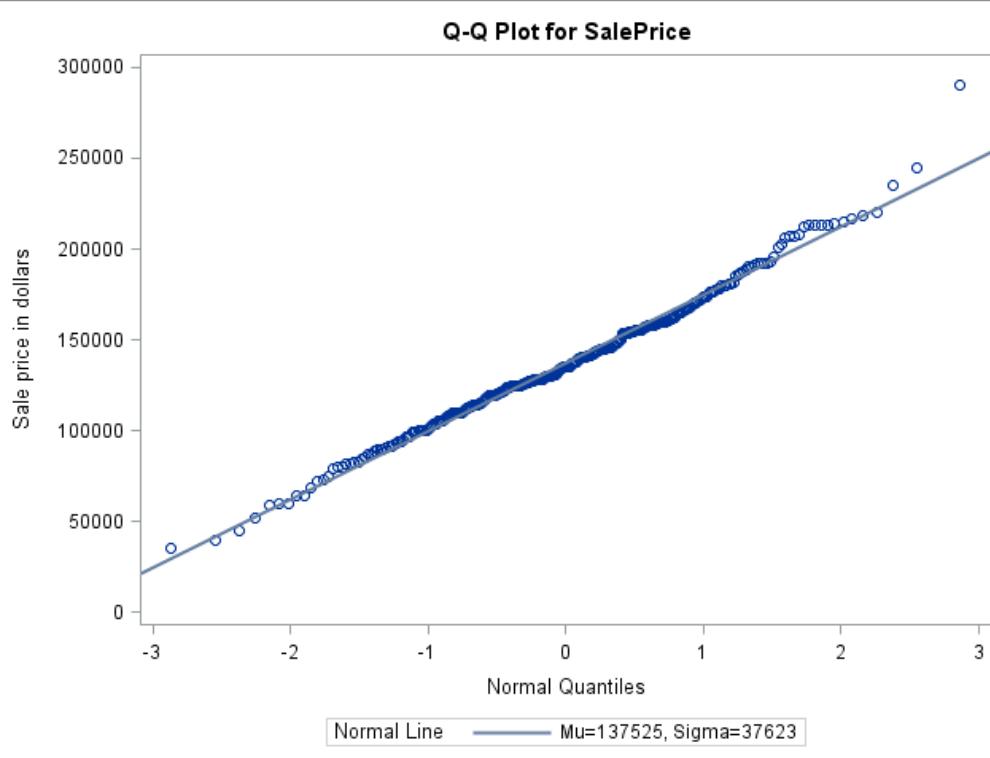
```
ods select basicmeasures qqplot;
proc univariate data=sp4r.ameshousing;
  var saleprice;
  qqplot saleprice / normal(mu=est sigma=est);
run;
```

```
The UNIVARIATE Procedure
Variable: SalePrice

Basic Statistical Measures

Location          Variability
Mean      137524.9      Std Deviation      37623
Median    135000.0      Variance        1415463276
Mode      110000.0      Range           255000
                  Interquartile Range   45475

Note: The mode displayed is the smallest of 2 modes with a count of 6.
```



End of Demonstration



ODS Tables and Graphics

1. Go to <http://support.sas.com/documentation/> and select **Syntax Lookup 9.4**.

KNOWLEDGE BASE / DOCUMENTATION

KNOWLEDGE BASE

- [Products & Solutions](#)
- [System Requirements](#)
- [Install Center](#)
- [Third-Party Software Reference](#)
- **Documentation**
 - [What's New in SAS](#)
 - [Product Index A-Z](#)
 - [SAS 9.4](#)
 - [SAS Analytical Products 14.1](#)
 - [SAS Analytical Products 13.2](#)
 - [SAS Analytical Products 13.1](#)
 - [SAS 9.3](#)
 - [SAS Analytical Products 12.1](#)
 - [SAS 9.2](#)
 - [Earlier SAS Releases](#)
- [Papers](#)
- [Samples & SAS Notes](#)
- [Focus Areas](#)

SAS Product Documentation

Starting Points

- ▶ [Product Index A-Z](#)
- ▶ [Programmer's Bookshelf](#)
- ▶ [What's New in SAS](#)
- ▶ [Documentation by Title](#)

Syntax Shortcuts

- ▶ [Syntax Lookup 9.4 | 9.3](#) **(circled)**
- ▶ [SAS Procedures by Name and Product 9.4 | 9.3 | 9.2](#)
- ▶ [SAS Language Elements by Name, Product, and Category 9.4 | 9.3 | 9.2](#)

Search

Release

Product

Display All topics Examples only Syntax only

Submit

2. Enter **Univariate** and select **UNIVARIATE procedure**.

Enter a term:

[UNIVARIATE procedure](#)

3. Select **UNIVARIATE Procedure: Modeling a Data Distribution**.

UNIVARIATE procedure

Choose a topic to view:

Base SAS 9.4 Procedures Guide: Statistical Procedures, Fourth Edition

[UNIVARIATE Procedure: Modeling a Data Distribution](#)

4. Select the **Details** tab and then click **ODS Table Names**.

Table 4.40: ODS Tables Produced with the PROC UNIVARIATE Statement		
ODS Table Name	Description	Option
BasicIntervals	confidence intervals for mean, standard deviation, variance	CIBASIC
BasicMeasures	measures of location and variability	default
ExtremeObs	extreme observations	default
ExtremeValues	extreme values	NEXTRVAL=
Frequencies	frequencies	FREQ
LocationCounts	counts used for sign test and signed rank test	LOCOUNT
MissingValues	missing values	default, if missing values exist
Modes	modes	MODES
Moments	sample moments	default
Plots	line printer plots	PLOTS
Quantiles	quantiles	default
RobustScale	robust measures of scale	ROBUSTSCALE
SSPlots	line printer side-by-side box plots	PLOTS (with BY statement)
TestsForLocation	tests for location	default
TestsForNormality	tests for normality	NORMALTEST
TrimmedMeans	trimmed means	TRIMMED=
WinsorizedMeans	Winsorized means	WINSORIZED=

5. Select the **Details** tab and then select **ODS Graphics**.

Table 4.42: ODS Graphics Produced by PROC UNIVARIATE		
ODS Graph Name	Plot Description	Statement
CDFPlot	cdf plot	CDFPLOT
Histogram	histogram	HISTOGRAM
PPPlot	P-P plot	PPPLOT
ProbPlot	probability plot	PROBPLOT
QQPlot	Q-Q plot	QQPLOT

The ODS Table Names and ODS Graphics page shows the names of tables and graphics for a procedure as an alternative to using the ODS TRACE statement. It also shows the option or statement that is used to create the table or plot.

End of Demonstration

Output Delivery System (ODS)

- Create new data tables from the PROC step results tables.
- Choose which statistics to include in the new data tables.



39

Output Delivery System (ODS)

Use the ODS OUTPUT statement to save PROC step results tables as SAS data tables.

```
ods output basicmeasures = SP_BasicMeasures;
  ODS OUTPUT output-object-name = data-set-name;
proc univariate data=ameshousing;
  var saleprice;
run;
```

The diagram shows a table titled "Basic Statistical Measures" with two columns: "Location" and "Variability". The "Location" column contains "Mean", "Median", and "Mode". The "Variability" column contains "Std Deviation", "Variance", "Range", and "Interquartile Range". Below the table is a blue curved arrow pointing to a grid icon with a red ball at the bottom right corner, representing a SAS data set.

Basic Statistical Measures		
Location	Variability	
Mean	137524.9	Std Deviation
Median	135000.0	Variance
Mode	110000.0	Range
		255000
		Interquartile Range
		45475

40

Multiple object names can be specified to create multiple data tables.

Use the ODS TRACE ON statement to use the same table names identified from the log.

Output Delivery System (ODS)

Use the OUTPUT statement to customize the new data table.

```
proc univariate data=ameshousing;
  var saleprice;
  output out=stats mean=sp_mean;
run;
```

```
OUTPUT OUT=new-data-set-name
keyword-1 = variable-name-1 ... keyword-n = variable-name-n;
```

Basic Statistical Measures		
Location	Variability	
Mean	137524.9	Std Deviation
Median	135000.0	Variance
Mode	110000.0	Range
		255000
		Interquartile Range
		45475



41

Multiple OUTPUT statements can be specified to create multiple data tables.

The OUTPUT statement enables the user to select which individual values from the results tables to place in the new data table. This avoids keeping unwanted statistics from default results tables.



Creating Data Tables with ODS

SP4R05d03.sas

Use ODS OUTPUT and the OUTPUT statement to practice creating data tables from PROC step results tables.

1. Use PROC UNIVARIATE to explore the **SalePrice** variable. Use ODS SELECT to print only the BasicMeasures table and use ODS OUTPUT to create a SAS data table from the results table. Print the new table to ensure that it is created.

```
ods select basicmeasures;
ods output basicmeasures = sp4r.SalePrice_BasicMeasures;
proc univariate data=sp4r.ameshousing;
  var saleprice;
run;
```

The UNIVARIATE Procedure
Variable: SalePrice

Basic Statistical Measures

Location	Variability
----------	-------------

Mean	137524.9	Std Deviation	37623
Median	135000.0	Variance	1415463276
Mode	110000.0	Range	255000
		Interquartile Range	45475

Note: The mode displayed is the smallest of 2 modes with a count of 6.

```
proc print data=sp4r.saleprice_basicmeasures;
run;
```

Obs	VarName	Measure	Loc		
			LocValue	VarMeasure	VarValue
1	SalePrice	Mean	137524.9	Std Deviation	37623
2	SalePrice	Median	135000.0	Variance	1415463276
3	SalePrice	Mode	110000.0	Range	255000
4	SalePrice	-		Interquartile Range	45475

2. Duplicate the Quantile() function in R using the OUTPUT statement in PROC UNIVARIATE. Identify the 40th, 45th, 50th, 55th, and 60th percentiles of the **SalePrice** variable.

In order to identify percentiles other than the default percentiles, you must use an OUTPUT statement and create a new data table.

```
proc univariate data=sp4r.ameshousing;
  var saleprice;
  output out=sp4r.stats mean=saleprice_mean pctlpts=40, 45, 50, 55, 60
        pctlpct=saleprice_;
run;

proc print data=sp4r.stats;
run;
```

Selected PROC UNIVARIATE options:

- OUT specifies the name of the new SAS data table.
- PCTLPTS specifies the percentiles to be calculated for the VAR statement variables.
- PCTLPRE specifies one or more prefixes for the name of the variable to be created followed by the percentile listed in the PCTLPTS option.

Obs	saleprice_mean	saleprice_40	saleprice_45	saleprice_50	saleprice_55	saleprice_60
1	137524.87	127250	129950	135000	140000	145000

3. Use PROC MEANS to create a new data table with the mean of the **SalePrice** variable and the median of the **Garage_Area** variable. Print the data table to ensure that it is created.

```
proc means data=sp4r.ameshousing;
  var saleprice garage_area;
  output out=sp4r.stats mean(saleprice)=sp_mean
median(garage_area)=ga_med;
run;

proc print data=sp4r.stats;
run;
```

Specify the variable in parentheses after the statistic option to specify which variable and summary statistic combinations to output.

Obs	_TYPE_	_FREQ_	sp_mean	ga_med
1	0	300	137524.86667	390

4. Use PROC MEANS to create a new data table with the mean and standard deviation of both the **SalePrice** variable and the **Garage_Area** variable. Use the AUTONAME option to enables SAS to name the statistics. Print the data table to ensure that it is created.

```
proc means data=sp4r.ameshousing;
  var saleprice garage_area;
  output out=sp4r.stats mean= std= / autoname;
run;

proc print data=sp4r.stats;
run;
```

Selected PROC MEANS option:

- AUTONAME specifies that PROC MEANS creates a unique variable name for an output statistic when you do not assign the variable name in the OUTPUT statement. This action is accomplished by appending the statistic-keyword to the end of the input variable name from which the statistic was derived.

Obs	_TYPE_	_FREQ_	SalePrice_Mean	Garage_Area_Mean	SalePrice_StdDev	Garage_Area_StdDev
1	0	300	137524.86667	369.45333333	37622.643128	176.25308979

5.3 Creating Macro Variables

Objectives

- Create user-defined macro variables with the %LET statement.
- Create user-defined macro variables with PROC SQL.

44

Motivation

Variables created in a DATA step cannot be used in subsequent DATA and PROC steps.

```
data myvars;  
  mymean = 123.45;  
  mysd = 49.6;  
run;
```

```
data test;  
  set test;  
  stan = (y-mymean)/mysd;  
run;
```

```
proc print data=test;  
  where y > mymean;  
run;
```

45

The variables **mymean** and **mysd** created in the **myvars** data set cannot be used as variables in other DATA or PROC steps that do not use the **myvars** data set.

Duplicating the R Script

```
#Create new variables  
mymean = 123.45  
mysd = 49.6  
stan = (y - mymean) / mysd  
  
#Automate the process  
mymean = mean(y)  
mysd = sd(y)  
stan = (y - mymean) / mysd
```

46

Motivation II

- Manually create a macro variable.
 - Automate creating a macro variable.



47

%LET Statement

Use the %LET statement to manually create a macro variable and assign a value to it.

- Create a macro variable from a numeric variable.



A screenshot of the SAS Studio interface. The code editor shows the following line of code:

```
height = 67;
```

The line `height = 67;` is highlighted with a blue selection bar. A tooltip window is displayed below the cursor, showing the syntax: `%LET variable-name = value;`

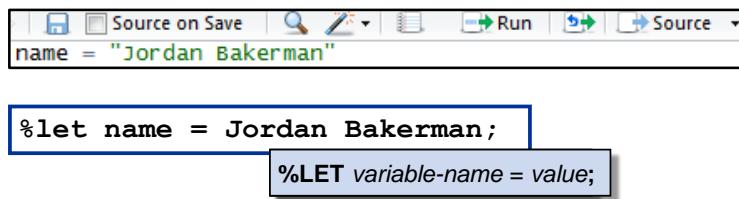
48

%LET statements are valid in open code, anywhere in a SAS program.

%LET Statement

Use the %LET statement to manually create a macro variable and assign a value to it.

- Create a macro variable from a character variable.



A screenshot of the SAS Studio interface. The code editor shows the following line of code:

```
name = "Jordan Bakerman";
```

The line `name = "Jordan Bakerman";` is highlighted with a blue selection bar. A tooltip window is displayed below the cursor, showing the syntax: `%LET variable-name = value;`

49

Macro variables are stored as text. Thus, the same syntax is used.

Macro Variables

Macro variables have the following characteristics.

- Number tokens are stored as text strings.
- The minimum length is 0 characters (null value).
- The maximum length is 64k characters.
- Case is preserved.
- Leading and trailing blanks are removed.
- Quotation marks are stored as part of the value.

50

Macro Variable References

Macro variable references begin with an ampersand (&) followed by a macro variable name.



Macro variable references

- are also called *symbolic references*
- can appear anywhere in a program
- are not case sensitive
- are passed to the macro processor.

51

5.03 Multiple Answer Poll

Which statements are true regarding macro variables?
(Select all that apply.)

- a. Macro variables must be assigned in a DATA or PROC step.
- b. Case is preserved.
- c. To reference a macro variable, you must use the & symbol.
- d. Macro variables can be used only three times or less in a PROC step.

52

%LET Statement Example I

Use a macro variable to code a numeric variable.

```
proc print data=ameshousing;
  where yr_sold = 2010;
  var yr_sold saleprice;
  title "Price of Homes Sold in 2010";
run;
```

Partial PROC PRINT Output

Price of Homes Sold in 2010		
Obs	Yr_Sold	SalePrice
1	2010	213500
2	2010	191500
3	2010	115000

54

%LET Statement Example I

Reference the macro variable with the & symbol.

```
%let year = 2010;

proc print data=ameshousing;
  where yr_sold = &year;
  var yr_sold saleprice;
  title "Price of Homes Sold in &year";
run;
```

Partial PROC PRINT Output

Price of Homes Sold in 2010		
Obs	Yr_Sold	SalePrice
1	2010	213500
2	2010	191500
3	2010	115000

55

Replace the year **2010** with a macro variable so that it can be changed easily.

-  You must use double quotation marks when you refer to a macro variable.

%LET Statement Example II

Use a macro variable to code a character variable.

```
proc print data=ameshousing;
  where garage_type_2 = "Attached";
  var yr_sold saleprice;
  title "Homes Sold with Attached Garage";
run;
```

Partial PROC PRINT Output

Homes Sold with Attached Garage		
Obs	Yr_Sold	SalePrice
1	2010	213500
2	2010	191500
4	2010	160000

56

%LET Statement Example II

Reference the macro variable with the & symbol.

```
%let gtype = Attached;

proc print data=ameshousing;
  where garage_type_2 = "&gtype";
  var yr_sold saleprice;
  title "Homes Sold with &gtype Garage";
run;
```

Partial PROC PRINT Output

Homes Sold with Attached Garage		
Obs	Yr_Sold	SalePrice
1	2010	213500
2	2010	191500
4	2010	160000

57

Replace the word **Attached** with a macro variable. SAS can easily exchange text because all macro variables are stored as text strings.

5.04 Poll

The SAS code below creates the PROC CORR and PROC MEANS analyses.

- True
- False

```
%let cont_var = saleprice garage_area
               basement_area gr_liv_area;

ods select pearsoncorr;
proc corr data=sp4r.ameshousing;
  var cont_var;
run;

proc means data=sp4r.ameshousing;
  var cont_var;
run;
```

58

Creating a Macro Variable in SQL

Step 1 Create a SAS data table.

```
proc means data=ameshousing;
  var saleprice;
  output out=stats mean=mean std=sd;
run;
```

PROC Step



60

First, create a SAS data table using the tools from Section 2.

Create a Macro Variable in SQL – Part II

Step 2 Use PROC SQL to create a macro variable.

```
proc sql;
  select mean into :sp_mean from stats;
  select sd into :sp_sd from stats;
quit;
```

```
PROC SQL;
  SELECT variable-name
  INTO :macro-variable-name
  FROM data-table-name;
```



PROC SQL



&

61

Second, use the new data table and PROC SQL to create macro variables.

PROC SQL can specify multiple SELECT statements to create multiple macro variables simultaneously.

- PROC SQL uses a QUIT statement to end the procedure.

%PUT Statement

The %PUT statement writes text to the SAS log.

```
%put The mean and sd of the Sale Price  
variable is &sp_mean and &sp_sd;
```

%PUT *text*;

SAS Log

```
The mean and sd of the Sale Price variable is  
137524.87 and 37622.64
```

- ✍ Quotation marks are not required around text.
- ✍ The %PUT statement is valid in open code (anywhere in a SAS program).

62

Displaying Macro Variables

The _USER_ argument in the %PUT statement writes the names and values of all user-defined macro variables to the SAS log.

%PUT _USER_;

SAS Log

```
135 %put _USER_;  
GLOBAL GTYPE Attached  
GLOBAL YEAR 2010  
GLOBAL SP_MEAN 137524.86667  
GLOBAL SP_SD 37622.643128
```

63

Use _ALL_ to display all user-defined and SAS macro variables in the SAS log.

Automatic Macro Variables

Automatic macro variables have fixed values, are created at SAS invocation, and are global in scope.

Name	Description
SYSDATE	Date of SAS invocation (06JAN14)
SYSDATE9	Date of SAS invocation (06JAN2014)
SYSDAY	Day of the week of SAS invocation (Friday)
SYSTIME	Time of SAS invocation (10:47).
SYSSCP	Operating system abbreviation (WIN, OS, HP 64)
SYSVER	Release of SAS software (9.3)
SYSUSERID	Login or user ID of current SAS process

-  The macro variables **SYSDATE**, **SYSDATE9**, and **SYSTIME** store text, not SAS date or time values.

To view all automatic macro variables go to the SAS documentation at
<http://support.sas.com/documentation/cdl/en/mcrolref/67912/HTML/default/viewer.htm#n18mk1d0g1j3lin1q6chazvfseel.htm>.



Creating and Using Macro Variables

SP4R05d04.sas

1. Create a new data table named **Stats** with the mean and standard deviation of the **SalePrice** variable. Then use PROC SQL to create two different macro variables for the two statistics in the **Stats** data table.

```
proc means data=sp4r.ameshousing;
  var saleprice;
  output out=sp4r.stats mean=sp_mean std=sp_sd;
run;

proc sql;
  select sp_mean into :sp_mean from sp4r.stats;
  select sp_sd into :sp_sd from sp4r.stats;
quit;
```

sp_mean	
<hr/>	
137524.86667	
sp_sd	
<hr/>	
37622.643128	

2. With the **AmesHousing** data set, use macro variables to create a new variable that represents the standardized values of **SalePrice**. Print the first six observations of the new standardized variable and **SalePrice**. Also, use PROC MEANS to print the mean and standard deviation for the two variables.

```
data sp4r.ameshousing;
  set sp4r.ameshousing;
  sp_stan = (saleprice - &sp_mean) / &sp_sd;
run;

proc print data=sp4r.ameshousing (obs=6);
  var saleprice sp_stan;
run;

proc means data=sp4r.ameshousing mean std;
  var saleprice sp_stan;
run;
```

Obs	SalePrice	sp_stan
1	213500	2.01940
2	191500	1.43464
3	115000	-0.59871
4	160000	0.59738
5	180000	1.12898
6	125000	-0.33291

The MEANS Procedure

Variable	Mean	Std Dev
SalePrice	137524.87	37622.64
sp_stan	-8.85989E-11	1.0000000

3. Use the OUT= option in PROC CONTENTS to save the output information in a SAS data table named **carscontents**. Then print the variables **name** and **type** from the new table.

```
proc contents data=sp4r.cars varnum out=carscontents;
run;

proc print data=carscontents;
  var name type;
run;
```

Obs	NAME	TYPE
1	Cylinders	1
2	DriveTrain	2
3	EngineSize	1
4	Horsepower	1
5	Invoice	1
6	Length	1
7	MPG_City	1
8	MPG_Highway	1
9	MSRP	1
10	Make	2
11	Model	2
12	Origin	2
13	Type	2
14	Weight	1
15	Wheelbase	1

 The **type** variable uses a numeric reference to indicate whether the data is numeric (1) or character (2).

4. Use PROC SQL to create two macro variables: **vars_cont**, which represents the continuous variables, and **vars_cat**, which represents the categorical variables. Then print the new macro variables inside a sentence using %PUT.

```
proc sql;
  select distinct name into: vars_cont separated by ' '
    from carscontents where type=1;
  select distinct name into: vars_cat  separated by ' '
    from carscontents where type=2;
quit;

%put The continuous variables are &vars_cont and the categorical
variables are &vars_cat;
```

Select PROC SQL options:

SELECT	initializes the query and selects the columns or rows.
DISTINCT	eliminates duplicate rows.
INTO:	stores the value in the subsequent macro variables.
SEPARATED BY	specifies to group the data by the subsequent method in single quotation marks.
FROM	specifies the data table to query.
WHERE	provides conditional arguments for the query.

Obs	NAME	TYPE
1	Cylinders	1
2	DriveTrain	2
3	EngineSize	1
4	Horsepower	1
5	Invoice	1
6	Length	1
7	MPG_City	1
8	MPG_Highway	1
9	MSRP	1
10	Make	2
11	Model	2
12	Origin	2
13	Type	2
14	Weight	1
15	Wheelbase	1

```
2886 %put The continuous variables are &vars_cont and the categorical variables are &vars_cat;
The continuous variables are Cylinders EngineSize Horsepower Invoice Length MPG_City MPG_Highway
MSRP Weight Wheelbase and the categorical variables are DriveTrain Make Model Origin Type
```

End of Demonstration

5.4 Creating Macro Programs

Objectives

- Create a macro using the %MACRO and %MEND statements.
- Supply the macro with parameters.
- Use macro statements for conditional processing.
- Use macro statements for iterative processing.

67

Motivation

- Create a macro program to use SAS code repetitively.
- Run SAS DATA and PROC step code conditionally and iteratively.



68

Duplicating the R Script

```
#Create a Function
randnorm = function(n){

  #Generate Data
  vec = rnorm(n)

  #Print Summary Statistics
  mean(vec)
  median(vec)
  sd(vec)
  min(vec)
  max(vec)

  #Create Plots
  par(mfrow=c(1,2))
  hist(vec)
  plot(vec,type="b")
}

#Use Function and Change Parameters
randnorm(n=10000)
```

69

Idea Exchange

What types of functions do you make in R that call R functions and packages?

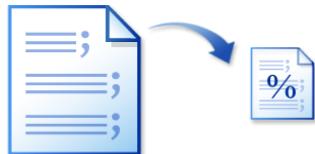
How can this be translated into SAS PROC and DATA steps?



70

Purpose of the Macro Facility

The *macro facility* is a ***text processing facility*** for automating and customizing the generation of SAS code, which then minimizes the amount of code that you must enter.



The macro facility supports the following:

- symbolic substitution within SAS code
- automated production of SAS code
- conditional construction of SAS code

71

Defining a Macro

Use the %MACRO and %MEND statements to define a macro. This code defines the **Today** macro, which displays the day and date.

```
%macro today;
  %put Today is &sysday &sysdate9;
%mend;
%MACRO macro-name;
  macro-text
%MEND <macro-name>;
```

- ✍ Macro names follow SAS naming conventions and cannot be reserved names such as names of macro statements or functions (for example, LET and SCAN).

72

After the macro is compiled, the macro is stored in the **Work** library with the name **sasmacr**.

Calling a Macro

To call a macro, precede the macro name with a percent sign (%).



SAS Log

```
178 %today
Today is Friday 01JAN2016
```

A *macro call*

- can appear anywhere (similar to a macro variable reference)
- is passed to the macro processor
- is **not** a statement (no semicolon required)
- causes the macro to execute.

73

Calling the time macro causes the %PUT statement to execute.

Parameter Lists

A *parameter list* is a list of macro variables referenced within the macro. There are three types of parameter lists:

- positional
- keyword
- mixed



74

Positional Parameters

Parameter **values** are supplied when the macro is called.

```
%MACRO macro-name(parameter-1, ... parameter-n);  
  
%macro calc(dsn,vars);  
  proc means data=&dsn;  
    var &vars;  
  run;  
%mend calc;  
  
%calc(business,yield)  
  
%macro-name(value-1, ... value-n)
```

- ✍ Parameter values must appear in the same order as their corresponding parameter names.

75

Keyword Parameters

Keyword parameters are assigned a default value after an equal sign (=).

```
%macro count(opts=,start=01jan08,stop=31dec08);  
  proc freq data=orion.orders;  
    where order_date between  
      "&start" and "&stop";  
    table order_type / &opts;  
    title1 "Orders from &start to &stop";  
  run;  
%mend count;  
  
%MACRO macro-name(keyword=value, ..., keyword=value);  
  macro text  
  %MEND <macro-name>;
```

76

The first parameter has a null value by default. On the final call, all parameters receive their default value. The empty parentheses are important because this macro “knows” that it has a parameter list. If you omit the parentheses, the macro does not execute but patiently waits for its expected parameter list. If the next token submitted does not begin a parameter list, the macro “knows” that a parameter list is not forthcoming and executes using default parameter values. Parentheses, even if empty, are recommended as explicit and unambiguous, and they guarantee immediate execution of the macro.

Keyword Parameters

Call a macro with keyword parameters.

```
%macro count(opts=,start=01jan08,stop=31dec08);
  proc freq data=orion.orders;
    where order_date between
      "&start" and "&stop";
    table order_type / &opts;
    title1 "Orders from &start to &stop";
  run;
%mend count;
options mprint;
%count(opts=nocum)
%count(stop=01jul08,opts=nocum nopercnt)
%count() %macro-name(keyword=value, ..., keyword=value)
```

Keyword parameters can appear in any order and can be omitted from the call without placeholders. If omitted from the call, a keyword parameter receives its default value.

77

Mixed Parameter Lists

In a mixed parameter list, positional parameters are listed before keyword parameters in both the macro definition and the macro call.

```
%macro count(opts,start=01jan08,stop=31dec08);
  proc freq data=orion.orders;
    where order_date between
      "&start" and "&stop";
    table order_type / &opts;
    title1 "Orders from &start to &stop";
  run;
%mend count;
options mprint;
%count(nocum)
%count(stop=30jun08,start=01apr08)
%count(nocum nopercnt,stop=30jun08)
%count()
```

78

5.05 Quiz

There are three mistakes in the SAS code below.
Can you find all three?

```
%macro test(condition=50000,dt);
  proc means data=dt;
    where msrp > &condition;
  run;
%mend;
%test(cars,100000)
```

79

Macro Statements

A macro language statement instructs the macro processor to perform an operation. It consists of a keyword, begins with the % symbol, and it ends in a semicolon.

Macro statements enable the following actions:

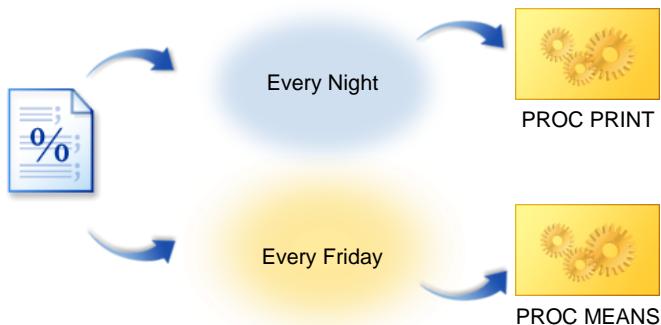
- conditional processing
- parameter validation
- iterative processing

81

Go to the SAS Help documentation to view a list of all the SAS macro statements.

Business Scenario

A daily sales report is generated every night. Every Friday, a weekly report is generated. Determine the best method to automate these reports.



82

Solutions

Two methods:

Method 1 Create multiple macros, including a driver macro.

Method 2 Create a single macro with %DO and %END statements.



83

Processing Complete Steps

Method 1 Create separate macros for the **Daily** and **Weekly** programs.

```
%macro daily;
  proc print data=orion.order_fact;
    where order_date="&sysdate9";
    var product_id total_retail_price;
    title "Daily sales: &sysdate9";
  run;
%mend daily;

%macro weekly;
  proc means data=orion.order_fact n sum mean;
    where order_date between
      "&sysdate9"-6 and "&sysdate9";
    var quantity total_retail_price;
    title "Weekly sales: &sysdate9";
  run;
%mend weekly;
```

84

Processing Complete Steps

Method 1 Create a driver macro that always calls the **Daily** macro and conditionally calls the **Weekly** macro.

```
%macro reports;
  %daily
  %if &sysday=Friday %then %weekly;
%mend reports;
```

**%IF expression %THEN action;
%ELSE action;**

- Character constants are
 - not quoted
 - case sensitive.
- The **%ELSE** statement is optional.
- **%IF-%THEN** and **%ELSE** statements can be used inside a macro definition only.

85

This is a true macro program, with a macro call and a macro language statement. This is a “system-driven” macro insofar as it is driven by or makes a decision according to system information, such as the day of the week.

Macro Expressions

%IF expression

	Macro Expressions	SAS Expressions
Arithmetic operators	✓	✓
Logical operators (Do not precede AND or OR with %.)	✓	✓
Comparison operators (symbols and mnemonics)	✓	✓
Case sensitivity	✓	✓
Special WHERE operators		
Quotation marks		✓
Ranges such as 1<=x<=10		✓
IN operator: parentheses required		✓

86

Processing Complete Steps

Method 2 Create a single macro with %DO and %END statements to generate text that contains semicolons.

```
%macro reports;
  proc print data=orion.order_fact;
    where order_date="%sysdate9";
    var product_id total_retail_price;
    title "Daily sales: &sysdate9";
  run;
%if &sysday=Friday %then %do;
  proc means data=orion.order_fact n sum mean;
    where order_date between
      "&sysdate9"d - 6 and "&sysdate9"d;
    var quantity total_retail_price;
    title "Weekly sales";
  run;
%end;
%mend reports;
```

%IF expression %THEN %DO;
statement; statement;...
%END;
%ELSE %DO;
statement; statement;...
%END;

87

Here is a second approach to the same application (Method 2). The advantage is that this is a single macro, not three macros as in Method 1. This is a common technique.

The %DO %END syntax enables the user to write multiple statements between the %DO and %END. This is useful for conditionally running DATA or PROC steps.



A Macro Program to Generate Data, Summary Statistics, and Plots

SP4R05d05.sas

1. Create a macro program called **mymac**. Give the macro two positional parameters (**dist** and **param1**) and four keyword parameters (**param2=**, **n=100**, **stats=no**, and **plot=no**).

```
%macro mymac(dist,param1,param2=,n=100,stats=no,plot=no);
```

2. Create conditions to terminate execution of the macro if the user does not supply a parameter for **dist** or **param1**. If they are not supplied, in the log, write a sentence that indicates which parameter is missing.

```
%if &dist= %then %do;
  %put Dist is a required argument;
  %return;
%end;

%if &param1= %then %do;
  %put Param1 is a required argument;
  %return;
%end;
```

Selected macro statements:

%IF %THEN conditionally process a portion of the macro.

%DO %END begins and ends a DO group that uses multiple statements.

%PUT writes text or macro variable information to the log.

%RETURN Execution causes normal termination of the currently executing macro.

3. Create conditions according to the number of parameters supplied by the user. Then, create a data set called **random**. The data set should have a randomly generated variable, **Y**, according to the parameters of the macro, and a sequence value **X**. The number of rows in the data table should be determined by the parameter **n**.

```
%if &param2= %then %do;
  data random (drop=i);
    do i=1 to &n;
      y=rand("&dist",&param1);
      x+1;
      output;
    end;
  run;
%end;

%else %do;
  data random (drop=i);
    do i=1 to &n;
      y=rand("&dist",&param1,&param2);
      x+1;
      output;
    end;
  run;
%end;
```

4. Create this condition. If the user supplies the parameter **stats=yes**, then the macro uses PROC MEANS to generate the mean and standard deviation for the variable **Y**. Use the %UPCASE function to ignore case sensitivity.

```
%if %upcase(&stats)=YES %then %do;
  proc means data=random mean std;
    var y;
  run;
%end;
```

5. Create another condition. If the user supplies the parameter **plot=yes**, then the macro uses the SGPLOT procedure to create a histogram and overlay a density plot for the variable **Y**. End the creation of the macro with the %MEND statement.

```
%if %upcase(&plot)=YES %then %do;
  proc sgplot data=random;
    histogram y / binwidth=1;
    density y / type=kernel;
  run;
%end;

%mend;
```

6. Call the macro without a distribution.

```
%mymac (param1=0.2,stats=yes)
```

```
2961 %mymac(param1=0.2,stats=yes)
Dist is a required argument
```

-  The %RETURN statement ends the execution of the macro because a distribution was not supplied.

7. Use the macro to create geometrically distributed random data with a parameter of 0.2. In addition, request that the PROC MEANS portion of the macro be called.

```
%mymac (dist=Geometric,param1=0.2,param2=,stats=yes)
```

The MEANS Procedure

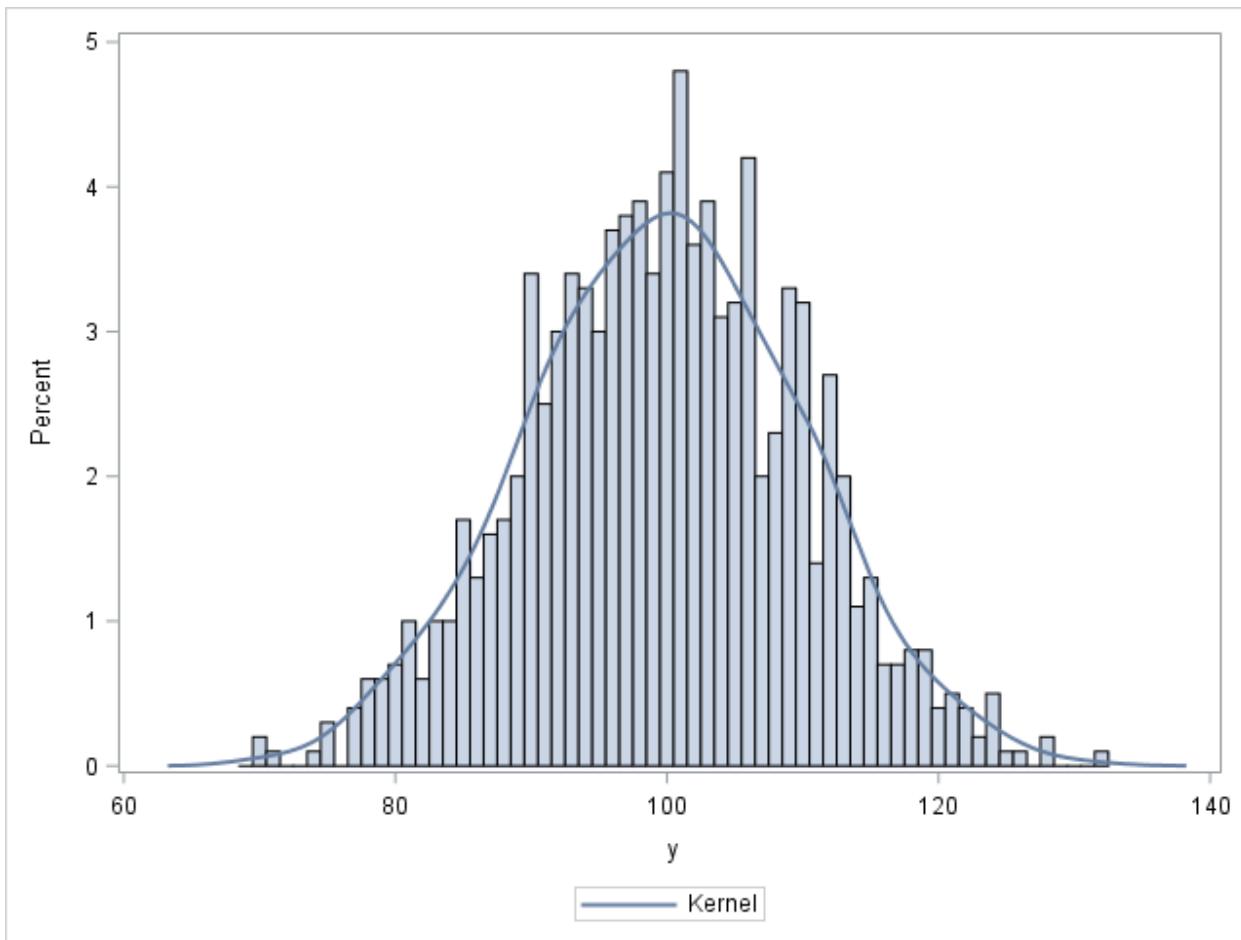
Analysis Variable : y

Mean	Std Dev
5.100000	4.6155206

8. Use the macro to create data distributed as $N(100,10)$. Generate 1000 observations and request that the plotting portion of the macro be called. In addition, use the MPRINT option to see what code is written to the log.

```
options mprint;
%mymac (dist=Normal,param1=100,param2=10,n=1000,plot=yes)
```

-  The MPRINT option is an excellent way to debug your macro variable. The MPRINT option displays the SAS statements that are generated by the macro execution.



```

993 /*Part H*/
994 options mprint;
995 %mymac(dist=Normal,param1=100,param2=10,n=1000,plot=yes)
MPRINT(MYMAC):   data random (drop=i);
MPRINT(MYMAC):   do i=1 to 1000;
MPRINT(MYMAC):     y=rand("Normal",100,10);
MPRINT(MYMAC):     x+1;
MPRINT(MYMAC):   output;
MPRINT(MYMAC):   end;
MPRINT(MYMAC):   run;

NOTE: The data set WORK.RANDOM has 1000 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds

MPRINT(MYMAC):   proc sgplot data=random;
MPRINT(MYMAC):     histogram y / binwidth=1;
MPRINT(MYMAC):     density y / type=kernel;
MPRINT(MYMAC):   run;

```

End of Demonstration



A Macro Program for Iterative Processing

SP4R05d06.sas

Example: A colleague saved the total sales for each of the company's 15 stores in CSV files separated by year (2000-2009). Read in each CSV file with a macro program that makes the iterative process quick and easy. Each file is named **sales_20##.csv** where ## refers to the year. Because each file has the same variable names, stack each file to create a SAS data set that stores all the company sales records.

1. Create a macro called **myappend**, which has two positional parameters (**start, stop**).

```
%macro myappend(start,stop);
```

2. Use the %DO, %TO, and %END macro statements to iterate through the positional parameters supplied by the user.

```
%do year=&start %to &stop;
```

3. Use PROC IMPORT to import each data set. Name each data set **sales_year** where **year** represents the numeric year corresponding to each CSV file.

```
proc import datafile="&path\sales_&year..csv"
            out=sp4r.sales_&year dbms=csv replace;
run;
```



Notice the double period notation in the data file path.

4. Use PROC APPEND to stack each newly created data set to create a single data set that holds all the information. Name this data set **sales_all**.

```
proc append base=sp4r.sales_all data=sp4r.sales_&year;
run;
```

5. Use PROC DATASETS to delete each data set after it is appended to **sales_all**. End both the %DO loop and the macro.

```
proc datasets library=sp4r noprint;
    delete sales_&year;
    quit;
%end;
%mend;
```

6. Call the macro and read in each CSV file by supplying the parameters **2000** and **2009**. Use the MPRINT option as a best practice.

```
options mprint;
%myappend(2000,2009)
```

7. Navigate to the **sales_all** data table in the **SP4R** library to view the data table. Notice that there are 150 observations, 15 for each year.
8. Why did the macro program above use two periods in the PROC IMPORT DATAFILE option?
A single period after a macro variable resolves that macro variable.

```
%let mypath = s:workshop\;
%put &mypathmydata.csv;
%put &mypath.mydata.csv;
```

```
5364 %let mypath = s:workshop\;
5365 %put &mypathmydata.csv;
WARNING: Apparent symbolic reference MYPATHMYDATA not resolved.
&mypathmydata.csv
5366 %put &mypath.mydata.csv;
s:workshop\mydata.csv
```

Notice that when you attempt to resolve **&mypathmydata.csv**, the macro processor tries to resolve the macro variable **mypathmydata**. To resolve the macro variable first, put a period after the macro variable.

9. What happens when you need to use a period to specify the file type? **Use two periods in this case, one to resolve the macro variable and one specify the file type.**

```
%let mydata = sales_data;
%put &mydata.csv;
%put &mydata..csv;
```

```
5367 %let mydata = sales_data;
5368 %put &mydata.csv;
sales_data.csv
5369 %put &mydata..csv;
sales_data.csv
```

End of Demonstration



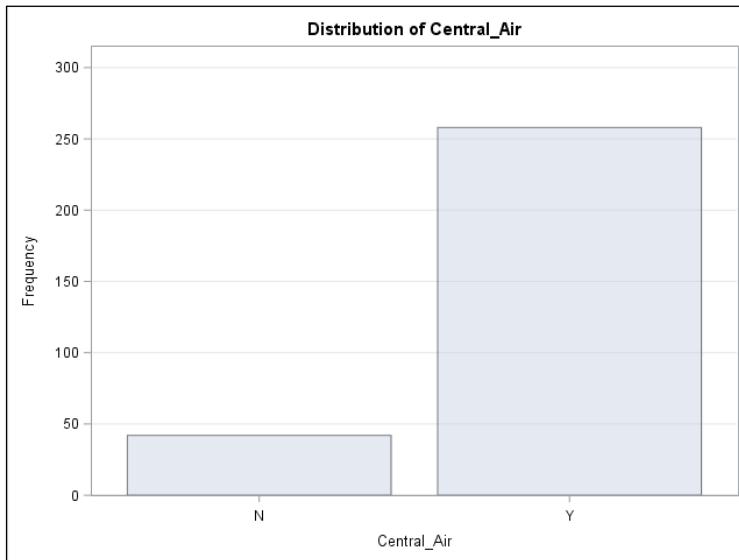
Exercises

Use the **AmesHousing** data set to complete Exercises 1, 2, 3, and 5. Use the **Cars** data set to complete Exercise 4.

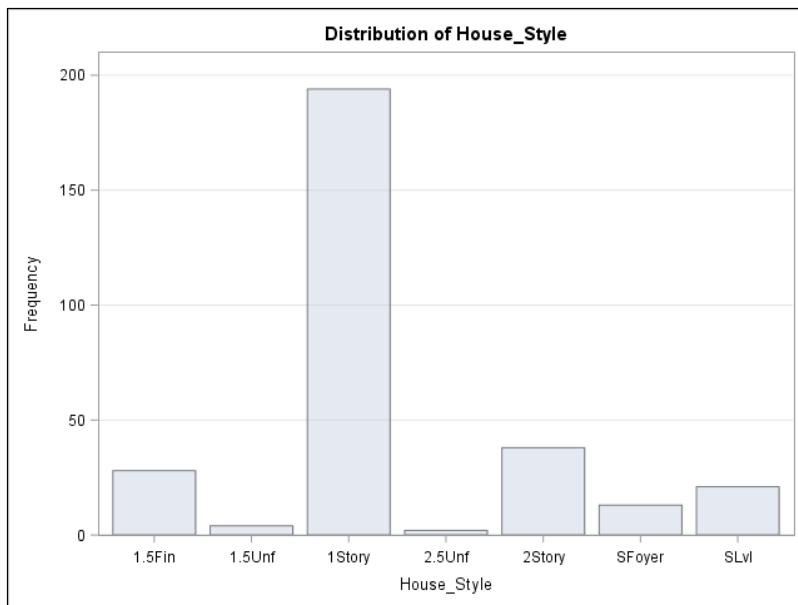
1. Using Descriptive Procedures and ODS

- a. Navigate to the SAS Help documentation and view the TABLES statement options for the FREQ procedure. Which option enables you to create a frequency plot? Use PROC FREQ to create one-way frequency tables for the variables **central_air** and **house_style** along with frequency plots. What percentage of homes in this sample have central air? What percent are only one story?

Central_Air	Frequency	Percent	Cumulative Frequency	Cumulative Percent
N	42	14.00	42	14.00
Y	258	86.00	300	100.00



House_Style	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1.5Fin	28	9.33	28	9.33
1.5Unf	4	1.33	32	10.67
1Story	194	64.67	226	75.33
2.5Unf	2	0.67	228	76.00
2Story	38	12.67	266	88.67
SFoyer	13	4.33	279	93.00
SLvl	21	7.00	300	100.00



- b. The default PROC CORR output gives a table of simple statistics and correlation coefficients. Use ODS SELECT to print only the correlation coefficients for the variables **saleprice**, **garage_area**, **basement_area**, and **gr_liv_area**. (Hint: It might be easiest to use the ODS TRACE statement to learn the table name instead of going to the documentation page.) Is there a statistically significant correlation between **saleprice** and each of the other variables?

Pearson Correlation Coefficients, N = 300				
Prob > r under H0: Rho=0				
	Sale Price	Garage_ Area	Basement_ Area	Gr_Liv_ Area
SalePrice	1.00000 <.0001	0.57892 <.0001	0.68956 <.0001	0.65046 <.0001
Garage_Area	0.57892 <.0001	1.00000 <.0001	0.35630 <.0001	0.33283 <.0001
Basement_Area	0.68956 <.0001	0.35630 <.0001	1.00000 <.0001	0.43985 <.0001
Gr_Liv_Area	0.65046 <.0001	0.33283 <.0001	0.43985 <.0001	1.00000 <.0001

- c. Use PROC MEANS to print the 10th percentile, median, and 90th percentile for the variables **saleprice** and **gr_liv_area**. In addition, use the CLASS statement to separate the summary statistics by the **yr_sold** variable. Finally, save the output using ODS OUTPUT and name the table **summary_table**. Print the table to ensure it is saved. Which year had the highest median sale price?

Obs	Yr_Sold	NObs	VName_SalePrice	SalePrice_P10	SalePrice_Median	SalePrice_P90
1	2006	55	SalePrice	93500	131000	169000
2	2007	72	SalePrice	96500	128500	180500
3	2008	62	SalePrice	87000	136250	181900
4	2009	73	SalePrice	91300	144000	192000

	5	2010	38	SalePrice	100000	148875	192000
Obs	VName_Gr_Liv_Area	Gr_Liv_Area_P10		Gr_Liv_Area_Median		Gr_Liv_Area_P90	
1	Gr_Liv_Area	864		1092		1368	
2	Gr_Liv_Area	864		1076		1435	
3	Gr_Liv_Area	864		1185		1430	
4	Gr_Liv_Area	800		1210		1456	
5	Gr_Liv_Area	848		1148.5		1395	

- d. Use PROC UNIVARIATE to analyze the **gr_liv_area** variable and create both a histogram and a Q-Q plot. For the histogram, overlay a normal and density kernel estimate. Use the OUTPUT statement to create a new data table of percentiles called **gr_perces**. Instead of providing the PCTLPTS= option a list, use the following syntax: **PCTLPTS= 40 to 60 by 2**. Let the prefixes for the saved percentiles be **gr_**. Print the table to ensure that it is saved.

Obs	gr_40	gr_42	gr_44	gr_46	gr_48	gr_50	gr_52	gr_54	gr_56	gr_58	gr_60
1	1063.5	1075.5	1087	1092	1109.5	1135	1151	1169.5	1191	1206	1218

2. Creating and Using a Macro Variable for Unsupervised Scripting

- Use the MEANS procedure to create a new data table with the median of the **SalePrice** variable.
- Use PROC SQL to create a macro variable of the median **SalePrice** value.
- In the **AmesHousing** data set, create a new variable that is a value of *1* if the **SalePrice** is greater than the median and *0* otherwise. Use PROC FREQ to create a frequency table of the new variable.

sp_bin	Frequency	Percent	Cumulative	Cumulative
			Frequency	Percent
0	153	51.00	153	51.00
1	147	49.00	300	100.00

3. Using the SYMPUTX Subroutine

- The SYMPUTX subroutine enables you to create a macro variable inside a DATA step. Navigate to the online documentation for a complete description. Run the SAS code below (**SP4R05e03.sas**) and analyze both the code and log output. What does this code do?

```
data _NULL_;
  x=-3;
  df=5;
  p=(1-probt(abs(x),df))*2;
  call symputx('sig_level',p);
run;

%put The significance level for the two-tailed t test is &sig_level;
```

- An alternative method to creating the macro variable in Exercise 2 is to use the SYMPUTX subroutine. Use a DATA _NULL_ step, a SET statement, and the SYMPUTX routine to create a macro variable for the median of the **saleprice** variable. Use the %PUT statement to ensure that the macro variable is created correctly.

4. Creating a Macro to Generate Summary Statistics and Plots of Any Data Table

- Open **SP4R05e04.sas**. Create the **mystats** macro. It should have a single positional parameter (**dt**) and four keyword parameters (**freq=no**, **means=no**, **opts=**, and **scatter=no**). Use the %IF, %THEN, and %END macro statements to validate the positional parameter. If no data table (**dt**) is supplied by the user, use %PUT to write the sentence “**dt** is a required argument” to the log and use the %RETURN statement to terminate the macro.
- Use PROC CONTENTS with the OUT= option to write the contents of the input data table (**dt**) to a new data table called **dtcontents**. Use PROC SQL to use the **Name** field from **dtcontents** to create two macro variables. Let **vars_cont** be the unique names of continuous variables in the data set separated by a space. Let **vars_cat** be the unique names of the categorical variables in the data set separated by a space.
Hint: Use the SAS code from the earlier demonstration in Chapter 5, **SP4R05d04**.
- Use macro statements to generate a PROC FREQ step if the user supplied **freq=yes** when calling **mystats**. In this case, use PROC FREQ to create frequency tables for the categorical variables.
- Use macro statements to generate a PROC MEANS step if the user supplies **means=yes**. In this case, specify the continuous variables in the VAR statement. In addition, use the **opts** parameter in the PROC MEANS statement to easily change the descriptive statistics.
- Use macro statements to set a condition if the user supplies **scatter=yes**. In this case, use PROC SGSCATTER to create a scatter plot matrix of the continuous variables. End the creation of the macro with %MEND.
- Call the **mystats** macro to create frequency tables for the **cars** data set.

Drive Train	Cumulative		Cumulative	
	Frequency	Percent	Frequency	Percent
All	92	21.50	92	21.50
Front	226	52.80	318	74.30
Rear	110	25.70	428	100.00
Make	Cumulative		Cumulative	
	Frequency	Percent	Frequency	Percent
Acura	7	1.64	7	1.64
Audi	19	4.44	26	6.07
BMW	20	4.67	46	10.75
Buick	9	2.10	55	12.85
Cadillac	8	1.87	63	14.72
Chevrolet	27	6.31	90	21.03
Chrysler	15	3.50	105	24.53
Dodge	13	3.04	118	27.57
Ford	23	5.37	141	32.94
GMC	8	1.87	149	34.81
Honda	17	3.97	166	38.79
Hummer	1	0.23	167	39.02
Hyundai	12	2.80	179	41.82
Infiniti	8	1.87	187	43.69
Isuzu	2	0.47	189	44.16
Jaguar	12	2.80	201	46.96
Jeep	3	0.70	204	47.66
Kia	11	2.57	215	50.23
Land Rover	3	0.70	218	50.93
Lexus	11	2.57	229	53.50

Lincoln	9	2.10	238	55.61
MINI	2	0.47	240	56.07
Mazda	11	2.57	251	58.64
Mercedes-Benz	26	6.07	277	64.72
Mercury	9	2.10	286	66.82
Mitsubishi	13	3.04	299	69.86
Nissan	17	3.97	316	73.83
Oldsmobile	3	0.70	319	74.53
Pontiac	11	2.57	330	77.10
Porsche	7	1.64	337	78.74
Saab	7	1.64	344	80.37
Saturn	8	1.87	352	82.24
Scion	2	0.47	354	82.71
Subaru	11	2.57	365	85.28
Suzuki	8	1.87	373	87.15
Toyota	28	6.54	401	93.69
Volkswagen	15	3.50	416	97.20
Volvo	12	2.80	428	100.00

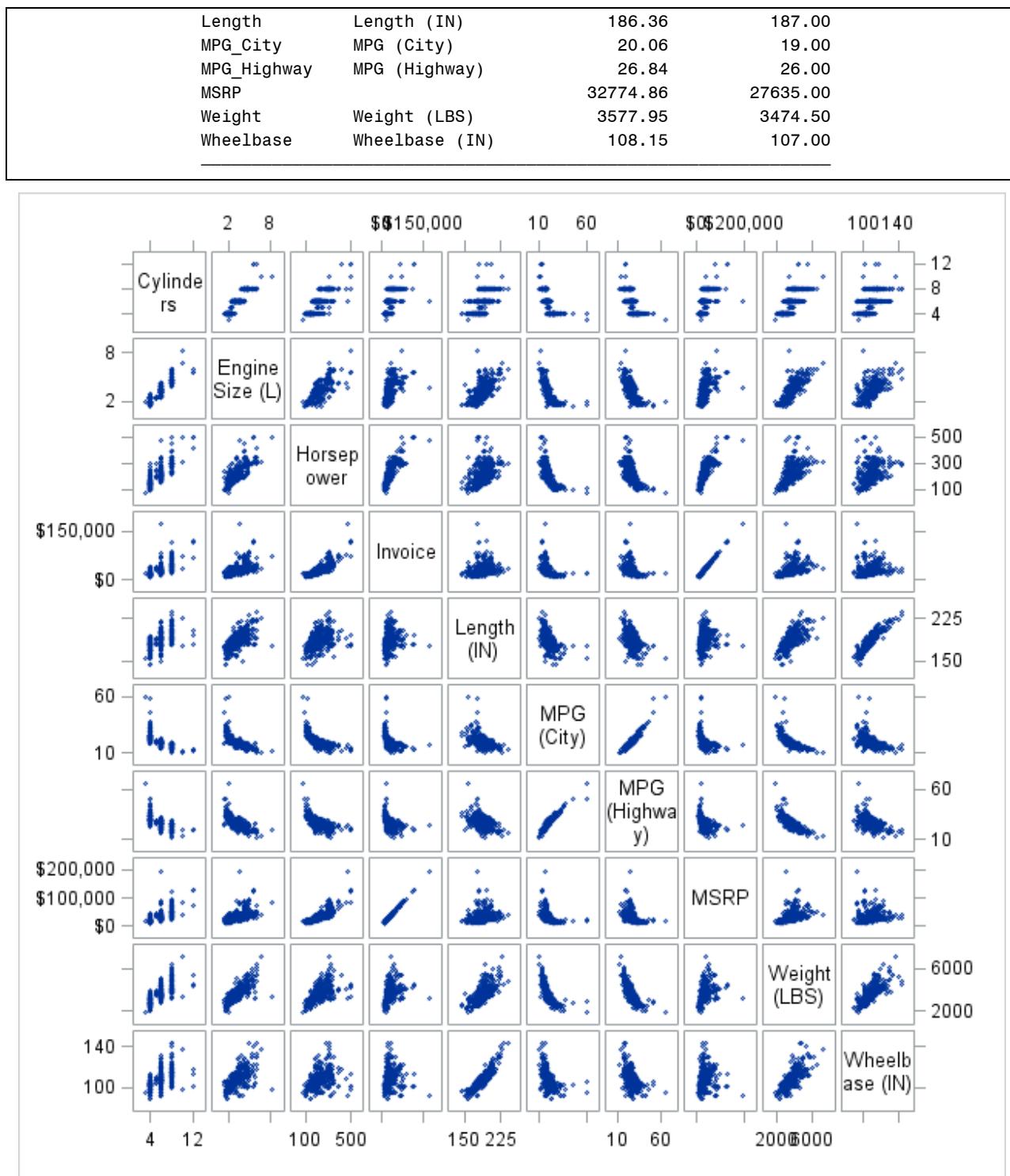
Partial Model Table

Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent
3.5 RL 4dr	1	0.23	1	0.23
3.5 RL w/Navigation 4dr	1	0.23	2	0.47
300M 4dr	1	0.23	3	0.70
300M Special Edition 4dr	1	0.23	4	0.93
325Ci 2dr	1	0.23	5	1.17

Origin	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Asia	158	36.92	158	36.92
Europe	123	28.74	281	65.65
USA	147	34.35	428	100.00
Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	0.70	3	0.70
SUV	60	14.02	63	14.72
Sedan	262	61.21	325	75.93
Sports	49	11.45	374	87.38
Truck	24	5.61	398	92.99
Wagon	30	7.01	428	100.00

- g. Call the **mystats** macro to create the means output with **opts=mean median maxdec=2**. Generate a scatter plot matrix for the continuous variables.

The MEANS Procedure				
Variable	Label		Mean	Median
Cylinders			5.81	6.00
EngineSize	Engine Size (L)		3.20	3.00
Horsepower			215.89	210.00
Invoice			30014.70	25294.50



 A macro variable created inside a macro program is *local in scope*. For example, **vars_cont** and **vars_cat** can be referenced only inside the macro program. To create a global macro variable, you must use the SYMPUTX subroutine. The third argument enables the user to specify a global option for the macro variable that is being created. View the SAS online documentation for a complete description.

5. Creating a Macro for Iterative Processing

Import a series of Excel workbook sheets into corresponding SAS data sets. The **amesbyyear** Excel workbook contains five separate sheets. Each sheet holds only the information for homes sold in a specific year. Each sheet is named according to the year (2006, 2007, 2008, 2009, 2010). The data begins on row 2, and row 1 contains all the variable names.

- a. Create a macro to iteratively call PROC IMPORT to read in each sheet of the **amesbyyear** spreadsheet. Call the macro **myimport** and give it two positional parameters (**firstyear**, **lastyear**). Let each new data set (one for each sheet) be named **year20##** where ## refers to each specific year.

 Remember that the iteration index value can be referenced as a macro variable.

- b. Call **myimport** to read in each sheet of the Excel file.
- c. Check the **SP4R** library to ensure that all five data sets are created.

End of Exercises

5.5 Solutions

Solutions to Exercises

The **AmesHousing** data set was used to complete Exercises 1, 2, 3, and 5. The **Cars** data set was used to complete Exercise 4.

1. Using Descriptive Procedures and ODS

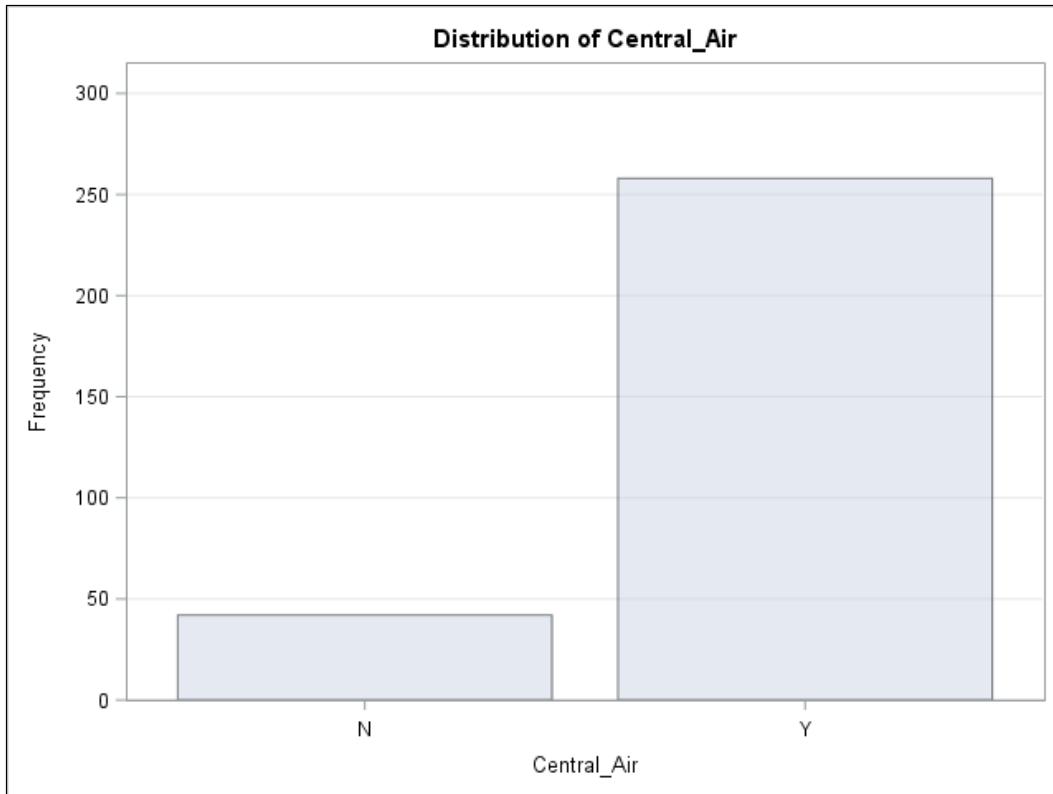
- Navigate to the SAS Help documentation and view the TABLES statement options for the FREQ procedure. Which option enables you to create a frequency plot? Use PROC FREQ to create one-way frequency tables for the variables **central_air** and **house_style** along with frequency plots. What percentage of homes in this sample have central air? What percent are only one story?

```
proc freq data=sp4r.ameshousing;
  tables central_air house_style / plots=freqplot;
run;
```

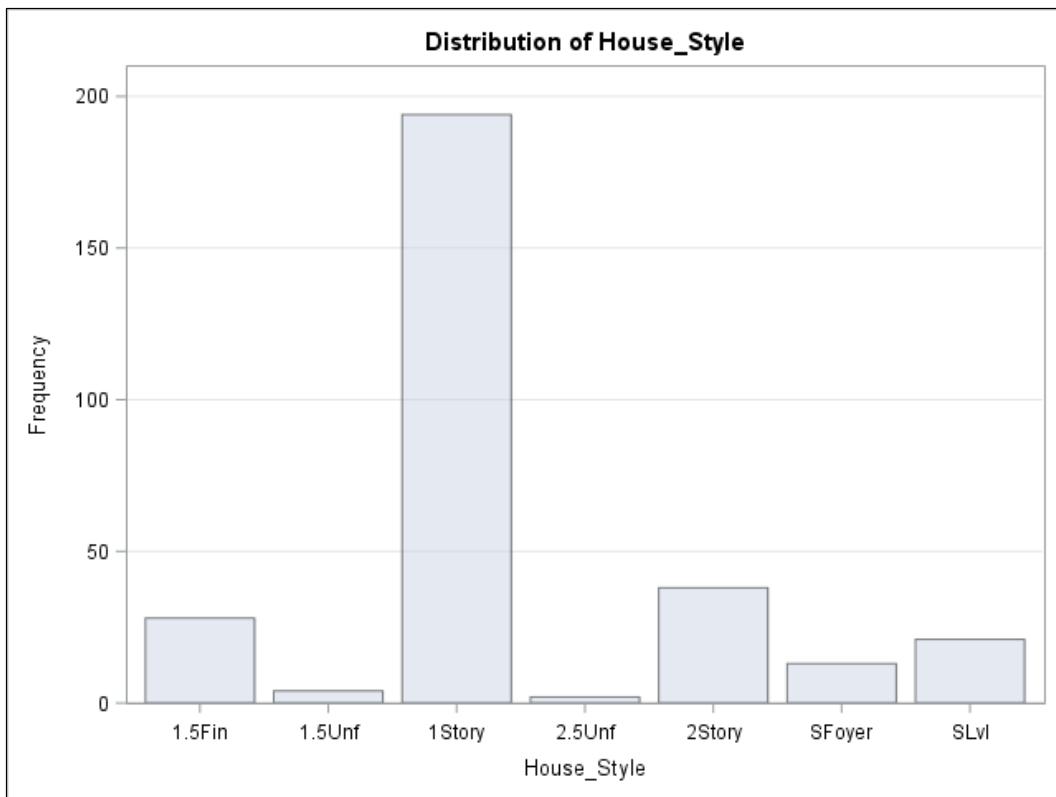
Selected PROC FREQ option:

PLOTS use the FREQPLOT option to display a frequency plot (bar chart) of the corresponding frequency table.

The FREQ Procedure				
Central_Air	Frequency	Percent	Cumulative Frequency	Cumulative Percent
N	42	14.00	42	14.00
Y	258	86.00	300	100.00



House_Style	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1.5Fin	28	9.33	28	9.33
1.5Unf	4	1.33	32	10.67
1Story	194	64.67	226	75.33
2.5Unf	2	0.67	228	76.00
2Story	38	12.67	266	88.67
SFoyer	13	4.33	279	93.00
SLvl	21	7.00	300	100.00



- b. The default PROC CORR output gives a table of simple statistics and correlation coefficients. Use ODS SELECT to print only the correlation coefficients for the variables **saleprice**, **garage_area**, **basement_area**, and **gr_liv_area**. (Hint: It might be easiest to use the ODS TRACE statement to learn the table name instead of going to the documentation page.) Is there a statistically significant correlation between **saleprice** and each of the other variables?

```
ods select pearsoncorr;
proc corr data=sp4r.ameshousing;
  var saleprice garage_area basement_area gr_liv_area;
run;
```

The CORR Procedure				
Pearson Correlation Coefficients, N = 300				
Prob > r under H0: Rho=0				
	Sale Price	Garage_ Area	Basement_ Area	Gr_Liv_ Area
SalePrice	1.00000	0.57892 <.0001	0.68956 <.0001	0.65046 <.0001
Garage_Area	0.57892 <.0001	1.00000	0.35630 <.0001	0.33283 <.0001
Basement_Area	0.68956 <.0001	0.35630 <.0001	1.00000	0.43985 <.0001
Gr_Liv_Area	0.65046 <.0001	0.33283 <.0001	0.43985 <.0001	1.00000

- c. Use PROC MEANS to print the 10th percentile, median, and 90th percentile for the variables **saleprice** and **gr_liv_area**. In addition, use the CLASS statement to separate the summary statistics by the **yr_sold** variable. Finally, save the output using ODS OUTPUT and name the table **summary_table**. Print the table to ensure it is saved. Which year had the highest median sale price?

```
ods output summary=summary_table;
proc means data=sp4r.ameshousing p10 median p90;
  var saleprice gr_liv_area;
  class yr_sold;
run;

proc print data=summary_table;
run;
```

Selected PROC MEANS statement:

CLASS specifies the variables whose values define the subgroup combinations for the analysis. Class variables are numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define levels of the variable.

Obs	Yr_Sold	NObs	VName_SalePrice	SalePrice_P10	SalePrice_Median	SalePrice_P90
1	2006	55	SalePrice	93500	131000	169000
2	2007	72	SalePrice	96500	128500	180500
3	2008	62	SalePrice	87000	136250	181900
4	2009	73	SalePrice	91300	144000	192000
5	2010	38	SalePrice	100000	148875	192000

Obs	VName_Gr_Liv_Area	Gr_Liv_Area_P10	Gr_Liv_Area_Median	Gr_Liv_Area_P90
1	Gr_Liv_Area	864	1092	1368
2	Gr_Liv_Area	864	1076	1435
3	Gr_Liv_Area	864	1185	1430
4	Gr_Liv_Area	800	1210	1456
5	Gr_Liv_Area	848	1148.5	1395

- d. Use PROC UNIVARIATE to analyze the **gr_liv_area** variable and create both a histogram and a QQPlot. For the histogram, overlay a normal and density kernel estimate. Use the OUTPUT statement to create a new data table of percentiles called **gr_percs**. Instead of providing the PCTLPTS= option a list, use the following syntax: **PCTLPTS= 40 to 60 by 2**. Let the prefixes for the saved percentiles be **gr_**. Print the table to ensure that it is saved.

```
proc univariate data=sp4r.ameshousing;
  var gr_liv_area;
  histogram gr_liv_area / normal kernel;
  qqplot gr_liv_area / normal(mu=est sigma=est);
  output out=gr_percs pctlpts= 40 to 60 by 2 pctlpre=gr_liv_area_;
run;

proc print data=gr_percs;
run;
```

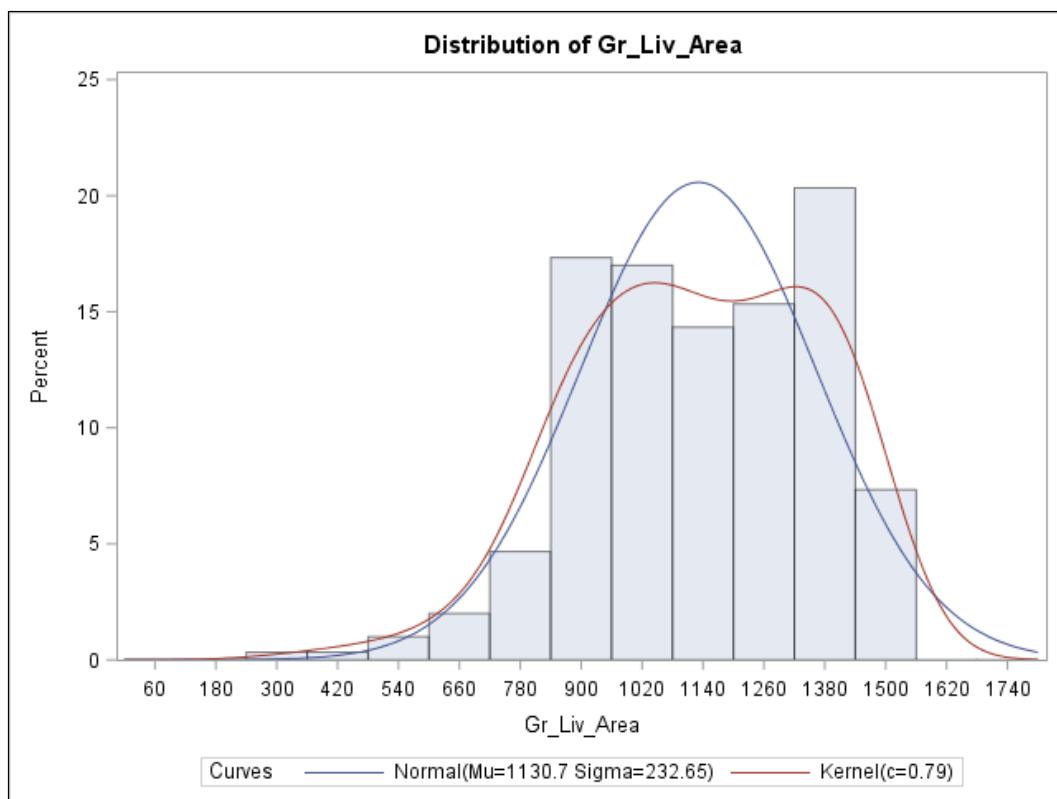
Selected PROC UNIVARIATE statements and options:

- OUT specifies the name of the new SAS data table.
- PCTLPTS specifies the percentiles to be calculated for the VAR statement variables.
- PCTLPRE specifies one or more prefixes for the name of the variable to be created followed by the percentile listed in the PCTLPTS option.
- VAR specifies numeric variables to analyze.
- HISTOGRAM specifies the numeric variable that is used to create a histogram. Use the NORMAL and KERNEL option to overlay a normal density and kernel density estimate.
- INSET specifies which statistics to include in the histogram plot. Use the POSITION= option to provide a location. Provide the option with a compass direction (NE = North East).
- QQPLOT specifies numeric variables to create a Q-Q plot. Use the NORMAL option to add a line to the Q-Q plot. Use the MU= and SIGMA= options to specify the parameters of the distribution for which quantiles are compared.

The UNIVARIATE Procedure			
Variable: Gr_Liv_Area			
Moments			
N	300	Sum Weights	300
Mean	1130.74	Sum Observations	339222
Std Deviation	232.649389	Variance	54125.7382
Skewness	-0.3905489	Kurtosis	-0.3328098
Uncorrected SS	399755480	Corrected SS	16183595.7
Coeff Variation	20.5749676	Std Error Mean	13.4320187
Basic Statistical Measures			
Location		Variability	
Mean	1130.740	Std Deviation	232.64939
Median	1135.000	Variance	54126
Mode	864.000	Range	1166
		Interquartile Range	385.50000
Tests for Location: Mu0=0			
Test	-Statistic-	-----p Value-----	
Student's t	t 84.18243	Pr > t	<.0001
Sign	M 150	Pr >= M	<.0001
Signed Rank	S 22575	Pr >= S	<.0001
Quantiles (Definition 5)			
Level		Quantile	
100% Max		1500.0	
99%		1490.0	
95%		1466.0	
90%		1431.0	
75% Q3		1337.5	

50% Median	1135.0
25% Q1	952.0
10%	847.0
5%	768.0
1%	509.0
0% Min	334.0

Extreme Observations			
-----Lowest-----		-----Highest---	
Value	Obs	Value	Obs
334	190	1484	142
438	100	1486	95
498	294	1494	181
520	145	1494	290
599	70	1500	222



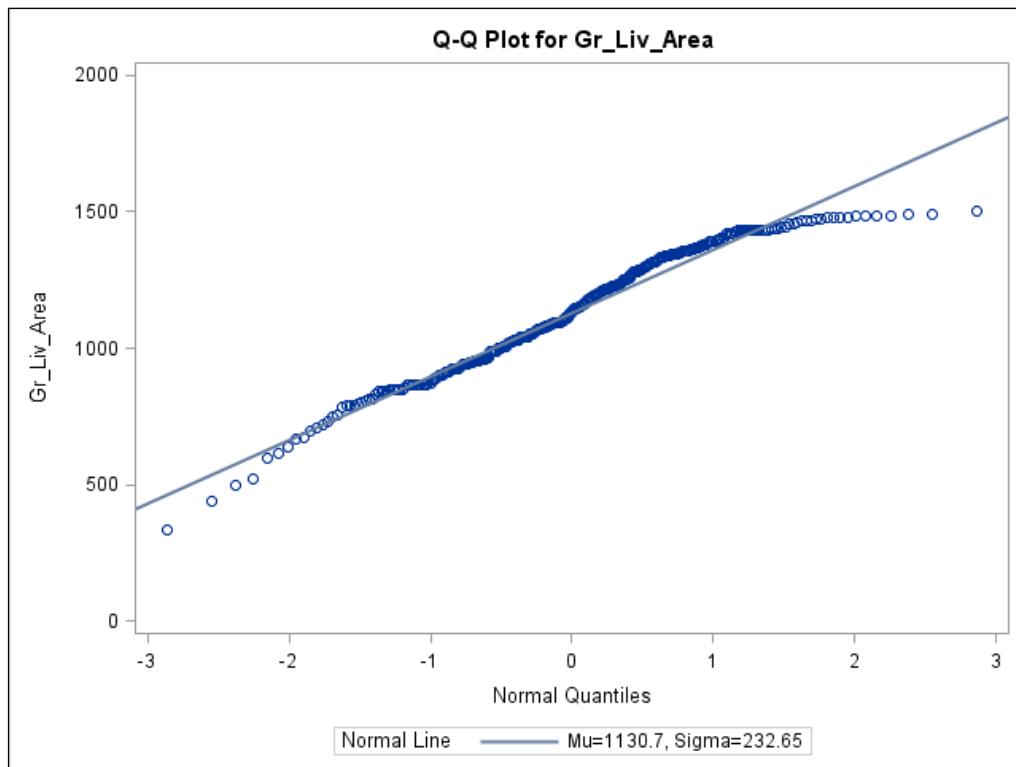
Fitted Normal Distribution for Gr_Liv_Area

Parameters for Normal Distribution

Parameter	Symbol	Estimate
Mean	μ	1130.74
Std Dev	σ	232.6494

Goodness-of-Fit Tests for Normal Distribution

Test	-----Statistic-----		-----p Value-----	
Kolmogorov-Smirnov	D	0.07537655	Pr > D	<0.010
Cramer-von Mises	W-Sq	0.33461417	Pr > W-Sq	<0.005
Anderson-Darling	A-Sq	2.41868951	Pr > A-Sq	<0.005
Quantiles for Normal Distribution				
Percent	-----Quantile-----			
	Observed	Estimated		
1.0	509.000	589.517		
5.0	768.000	748.066		
10.0	847.000	832.588		
25.0	952.000	973.820		
50.0	1135.000	1130.740		
75.0	1337.500	1287.660		
90.0	1431.000	1428.892		
95.0	1466.000	1513.414		
99.0	1490.000	1671.963		



Obs	gr_40	gr_42	gr_44	gr_46	gr_48	gr_50	gr_52	gr_54	gr_56	gr_58	gr_60
1	1063.5	1075.5	1087	1092	1109.5	1135	1151	1169.5	1191	1206	1218

2. Creating and Using a Macro Variable for Unsupervised Scripting

- a. Use the MEANS procedure to create a new data table with the median of the **SalePrice** variable.

```
proc means data=sp4r.ameshousing;
  var saleprice;
```

```
output out=sp4r.stats median=sp_med;
run;
```

- b. Use PROC SQL to create a macro variable of the median **SalePrice** value.

```
proc sql;
  select sp_med into :sp_med from sp4r.stats;
quit;
```

sp_med
135000

- c. In the **AmesHousing** data set, create a new variable that is a value of *I* if the **SalePrice** is greater than the median and *0* otherwise. Use PROC FREQ to create a frequency table of the new variable.

```
data sp4r.ameshousing;
  set sp4r.ameshousing;
  if saleprice > &sp_med then sp_bin = 1;
  else sp_bin = 0;
run;

proc freq data=sp4r.ameshousing;
  tables sp_bin;
run;
```

The FREQ Procedure

sp_bin	Frequency	Percent	Cumulative	Cumulative
			Frequency	Percent
0	153	51.00	153	51.00
1	147	49.00	300	100.00

3. Using the SYMPUTX Subroutine

- a. The SYMPUTX subroutine enables you to create a macro variable inside a DATA step. Navigate to the online documentation for a complete description. Run the SAS code below (**SP4R05e03.sas**) and analyze both the code and log output. What does this code do?

```
data _NULL_;
  x=-3;
  df=5;
  p=(1-probt(abs(x),df))*2;
  call symputx('sig_level',p);
run;

%put The significance level for the two-tailed t test is &sig_level;
```

The significance level for the two-tailed t test is 0.0300992479
--

This code uses a DATA _NULL_ step to create a macro variable for the significance level of a two-sided *t* test with five degrees of freedom and a test value of -3.

Selected functions and subroutines:

PROBT(x,df) returns the probability that an observation from a Student's *t* distribution, with degrees of freedom **df**, is less than or equal to **x**.

SYMPUTX assigns a value to a macro variable and removes both leading and trailing blanks.

- b. An alternative method to creating the macro variable in Exercise 2 is to use the SYMPUTX subroutine. Use a DATA _NULL_ step, a SET statement, and the SYMPUTX routine to create a macro variable for the median of the **saleprice** variable. Use the %PUT statement to ensure that the macro variable is created correctly.

```
proc means data=sp4r.ameshousing;
  var saleprice;
  output out=stats median=sp_med;
run;

data _null_;
  set stats;
  call symputx('med',sp_med);
run;

%put The median of the Sale Price variable is &med;
```

The median of the Sale Price variable is 135000

4. Creating a Macro to Generate Summary Statistics and Plots of Any Data Table

- a. Open SP4R05e04.sas. Create the **mystats** macro. It should have a single positional parameter (**dt**) and four keyword parameters (**freq=no**, **means=no**, **opts=**, and **scatter=no**). Use the %IF, %THEN, and %END macro statements to validate the positional parameter. If no data table (**dt**) is supplied by the user, use %PUT to write the sentence "dt is a required argument" to the log and use the %RETURN statement to terminate the macro.

```
%macro mystats(dt,freq=no,corr=no,means=no,opts=,scatter=no);

%if &dt= %then %do;
  %put dt is a required argument;
  %return;
%end;
```

- b. Use PROC CONTENTS with the OUT= option to write the contents of the input data table (**dt**) to a new data table called **dtcontents**. Use PROC SQL to use the **Name** field from **dtcontents** to create two macro variables. Let **vars_cont** be the unique names of continuous variables in the data set separated by a space. Let **vars_cat** be the unique names of the categorical variables in the data set separated by a space.

Hint: Use the SAS code from the earlier demonstration in Chapter 5, **SP4R05d04**.

```
proc contents data=&dt varnum out=dtcontents;
run;

proc sql;
  select distinct name into: vars_cont separated by ' '
    from dtcontents where type=1;
  select distinct NAME into: vars_cat  separated by ' '
    from dtcontents where type=2;
quit;
```

- c. Use macro statements to generate a PROC FREQ step if the user supplied `freq=yes` when calling `mystats`. In this case, use PROC FREQ to create frequency tables for the categorical variables.

```
%if %upcase(&freq)=YES %then %do;
  proc freq data=&dt;
    tables &vars_cat;
  run;
%end;
```

- d. Use macro statements to generate a PROC MEANS step if the user supplies `means=yes`. In this case, specify the continuous variables in the VAR statement. In addition, use the `opts` parameter in the PROC MEANS statement to easily change the descriptive statistics.

```
%if %upcase(&means)=YES %then %do;
  proc means data=&dt &opts;
    var &vars_cont;
  run;
%end;
```

- e. Use macro statements to set a condition if the user supplies `scatter=yes`. In this case, use PROC SGSCATTER to create a scatter plot matrix of the continuous variables. End the creation of the macro with %MEND.

```
%if %upcase(&scatter)=YES %then %do;
  proc sgscatter data=&dt;
    matrix &vars_cont;
  run;
%end;
%mend;
```

- f. Call the `mystats` macro to create frequency tables for the `cars` data set.

```
%mystats(sp4r.cars,freq=yes)
```

The FREQ Procedure

Drive Train	Frequency	Percent	Cumulative Frequency	Cumulative Percent
All	92	21.50	92	21.50
Front	226	52.80	318	74.30
Rear	110	25.70	428	100.00

Make	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Acura	7	1.64	7	1.64
Audi	19	4.44	26	6.07
BMW	20	4.67	46	10.75
Buick	9	2.10	55	12.85
Cadillac	8	1.87	63	14.72
Chevrolet	27	6.31	90	21.03
Chrysler	15	3.50	105	24.53
Dodge	13	3.04	118	27.57
Ford	23	5.37	141	32.94
GMC	8	1.87	149	34.81
Honda	17	3.97	166	38.79

Hummer	1	0.23	167	39.02
Hyundai	12	2.80	179	41.82
Infiniti	8	1.87	187	43.69
Isuzu	2	0.47	189	44.16
Jaguar	12	2.80	201	46.96
Jeep	3	0.70	204	47.66
Kia	11	2.57	215	50.23
Land Rover	3	0.70	218	50.93
Lexus	11	2.57	229	53.50
Lincoln	9	2.10	238	55.61
MINI	2	0.47	240	56.07
Mazda	11	2.57	251	58.64
Mercedes-Benz	26	6.07	277	64.72
Mercury	9	2.10	286	66.82
Mitsubishi	13	3.04	299	69.86
Nissan	17	3.97	316	73.83
Oldsmobile	3	0.70	319	74.53
Pontiac	11	2.57	330	77.10
Porsche	7	1.64	337	78.74
Saab	7	1.64	344	80.37
Saturn	8	1.87	352	82.24
Scion	2	0.47	354	82.71
Subaru	11	2.57	365	85.28
Suzuki	8	1.87	373	87.15
Toyota	28	6.54	401	93.69
Volkswagen	15	3.50	416	97.20
Volvo	12	2.80	428	100.00

Partial Model Table

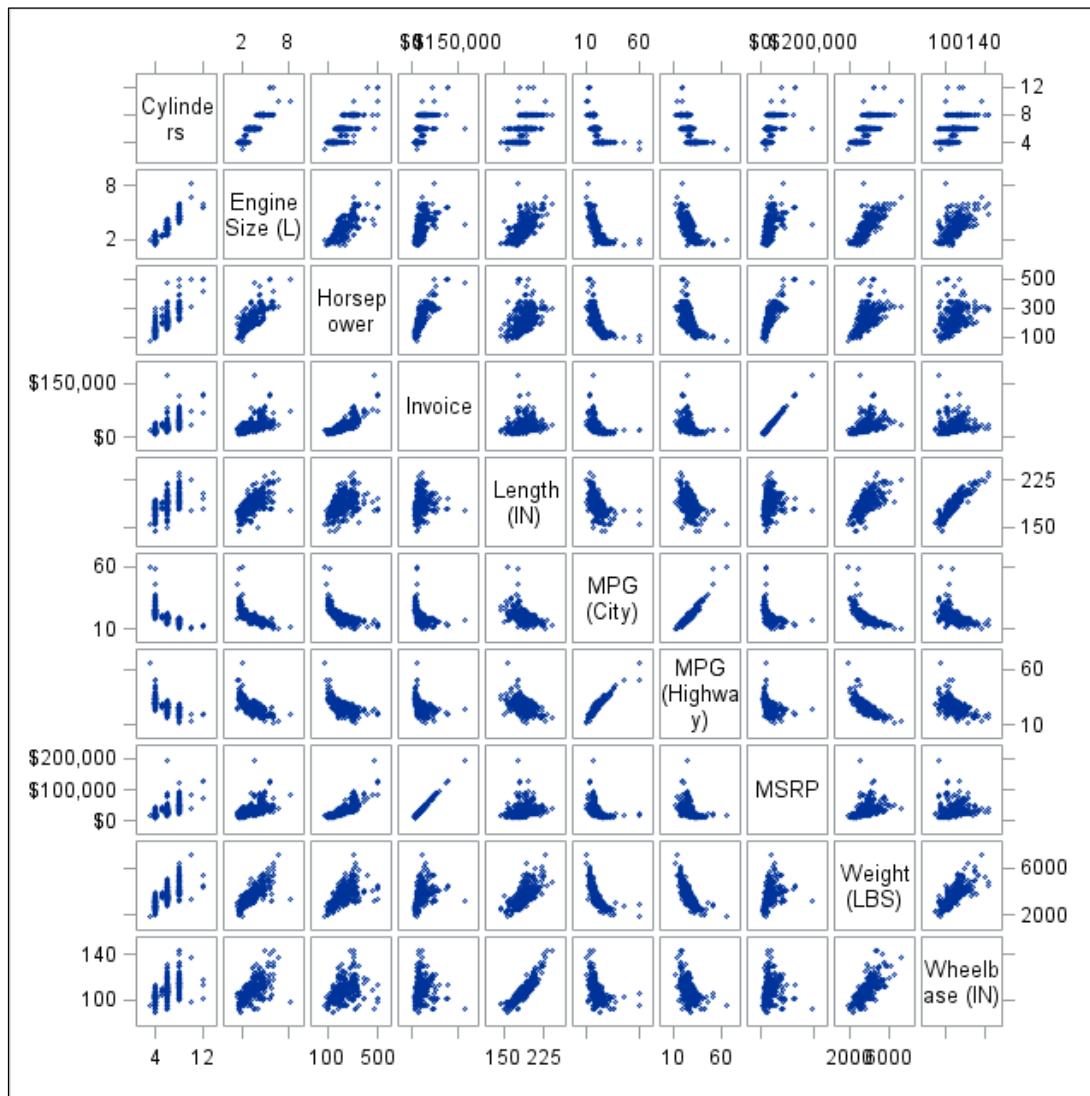
The FREQ Procedure					
Model	Frequency	Percent	Cumulative Frequency	Cumulative Percent	
3.5 RL 4dr	1	0.23	1	0.23	
3.5 RL w/Navigation 4dr	1	0.23	2	0.47	
300M 4dr	1	0.23	3	0.70	
300M Special Edition 4dr	1	0.23	4	0.93	
325Ci 2dr	1	0.23	5	1.17	
325Ci convertible 2dr	1	0.23	6	1.40	

Origin	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Asia	158	36.92	158	36.92
Europe	123	28.74	281	65.65
USA	147	34.35	428	100.00
Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	0.70	3	0.70
SUV	60	14.02	63	14.72
Sedan	262	61.21	325	75.93
Sports	49	11.45	374	87.38
Truck	24	5.61	398	92.99
Wagon	30	7.01	428	100.00

- g. Call the **mystats** macro to create the means output with **opts=mean median maxdec=2**.
 Generate a scatter plot matrix for the continuous variables.

```
%mystats(sp4r.cars,means=yes,opts=mean median maxdec=2,scatter=yes)
```

The MEANS Procedure				
Variable	Label		Mean	Median
Cylinders			5.81	6.00
EngineSize	Engine Size (L)		3.20	3.00
Horsepower			215.89	210.00
Invoice			30014.70	25294.50
Length	Length (IN)		186.36	187.00
MPG_City	MPG (City)		20.06	19.00
MPG_Highway	MPG (Highway)		26.84	26.00
MSRP			32774.86	27635.00
Weight	Weight (LBS)		3577.95	3474.50
Wheelbase	Wheelbase (IN)		108.15	107.00



5. Creating a Macro for Iterative Processing

Import a series of Excel workbook sheets into corresponding SAS data sets. The **amesbyyear** Excel workbook contains five separate sheets. Each sheet holds only the information for homes sold in a specific year. Each sheet is named according to the year (2006, 2007, 2008, 2009, 2010). The data begins on row 2, and row 1 contains all the variable names.

- Create a macro to iteratively call PROC IMPORT to read in each sheet of the **amesbyyear** spreadsheet. Call the macro **myimport** and give it two positional parameters (**firstyear**, **lastyear**). Let each new data set (one for each sheet) be named **year20##** where ## refers to each specific year.

 Remember that the iteration index value can be referenced as a macro variable.

```
%macro myimport(firstyear,lastyear);
  %do i=&firstyear %to &lastyear;
    proc import datafile = "&path\amesbyyear.xlsx"
      out = sp4r.year&i
      dbms = xlsx REPLACE;
      getnames = yes;
      sheet = "&i";
      datarow = 2;
    run;
  %end;
%mend;
```

- Call **myimport** to read in each sheet of the Excel file.

```
options mprint;
%myimport(2006,2010)
```

```
5078 options mprint;
5079 %myimport(2006,2010)
MPRINT(MYIMPORT): proc import out = year2006 datafile =
"C:\Users\jobake\Desktop\sp4rtest\amesbyyear.xlsx" dbms = xlsx REPLACE;
MPRINT(MYIMPORT):   RXLX;
MPRINT(MYIMPORT):   getnames = yes;
MPRINT(MYIMPORT):   sheet = "2006";
MPRINT(MYIMPORT):   datarow = 2;
MPRINT(MYIMPORT):   run;

MPRINT(MYIMPORT): proc import out = year2007 datafile =
"C:\Users\jobake\Desktop\sp4rtest\amesbyyear.xlsx" dbms = xlsx REPLACE;
MPRINT(MYIMPORT):   RXLX;
MPRINT(MYIMPORT):   getnames = yes;
MPRINT(MYIMPORT):   sheet = "2007";
MPRINT(MYIMPORT):   datarow = 2;
MPRINT(MYIMPORT):   run;

MPRINT(MYIMPORT): proc import out = year2008 datafile =
"C:\Users\jobake\Desktop\sp4rtest\amesbyyear.xlsx" dbms = xlsx REPLACE;
MPRINT(MYIMPORT):   RXLX;
MPRINT(MYIMPORT):   getnames = yes;
MPRINT(MYIMPORT):   sheet = "2008";
MPRINT(MYIMPORT):   datarow = 2;
MPRINT(MYIMPORT):   run;
```

```
MPRINT(MYIMPORT): proc import out = year2009 datafile =
"C:\Users\jobake\Desktop\sp4rttest\amesbyyear.xlsx" dbms = xlsx REPLACE;
MPRINT(MYIMPORT): RXLX;
MPRINT(MYIMPORT): getnames = yes;
MPRINT(MYIMPORT): sheet = "2009";
MPRINT(MYIMPORT): datarow = 2;
MPRINT(MYIMPORT): run;

MPRINT(MYIMPORT): proc import out = year2010 datafile =
"C:\Users\jobake\Desktop\sp4rttest\amesbyyear.xlsx" dbms = xlsx REPLACE;
MPRINT(MYIMPORT): RXLX;
MPRINT(MYIMPORT): getnames = yes;
MPRINT(MYIMPORT): sheet = "2010";
MPRINT(MYIMPORT): datarow = 2;
MPRINT(MYIMPORT): run;
```

- c. Check the SP4R library to ensure that all five data sets are created.

End of Solutions

Solutions to Student Activities (Polls/Quizzes)

5.01 Quiz – Correct Answer

Navigate to the TABLES statement in the SAS documentation page for the FREQ procedure.
What does the PROC step below do?

```
proc freq data=sp4r.cars;
  tables origin*type / chisq;
run;
```

Statistic	DF	Value	Prob
Chi-Square	10	35.6659	<.0001
Likelihood Ratio Chi-Square	10	42.1254	<.0001
Mantel-Haenszel Chi-Square	1	0.0808	0.7762
Phi Coefficient		0.2887	
Contingency Coefficient		0.2773	
Cramer's V		0.2041	

18

The CHISQ option requests chi-square tests of homogeneity of independence and measures of association that are based on the chi-square statistic.

5.02 Multiple Choice Poll – Correct Answer

What SAS procedures are used to reproduce the R functions min(), cov(), table(), and sd()?

- a. FREQ, MEANS, CORR, MEANS
- b. MEANS, CORR, FREQ, MEANS**
- c. MEANS FREQ, CORR, MEANS
- d. CORR, FREQ, FREQ, MEANS

24

5.03 Multiple Answer Poll – Correct Answers

Which statements are true regarding macro variables?
(Select all that apply.)

- a. Macro variables must be assigned in a DATA or PROC step.
- b. Case is preserved.
- c. To reference a macro variable, you must use the & symbol.
- d. Macro variables can be used only three times or less in a PROC step.

53

5.04 Poll – Correct Answer

The SAS code below creates the PROC CORR and PROC MEANS analyses.

- True
- False

```
%let cont_var = saleprice garage_area  
                basement_area gr_liv_area;  
  
ods select pearsoncorr;  
proc corr data=sp4r.ameshousing;  
    var cont_var;  
run;  
  
proc means data=sp4r.ameshousing;  
    var cont_var;  
run;
```

59

5.05 Quiz – Correct Answer

There are three mistakes in the SAS code below.
Can you find all three?

```
%macro test(dt, condition=50000);
  proc means data=&dt;
    where msrp > &condition;
  run;
%mend;
%test(cars, condition=100000)
```

80

- The positional parameters must come before the keyword parameters.
- You need to reference each parameter with the & symbol.
- The keyword parameters must use the parameter name when calling the macro.

Chapter 6 Analyzing the Data via Inferential Procedures

6.1 Linear Models.....	6-3
Demonstration: Multiple Linear Regression	6-10
Demonstration: Polynomial Regression	6-15
Demonstration: Two-Way ANOVA.....	6-27
Demonstration: ANCOVA.....	6-32
Demonstration: Stepwise Selection with PROC GLMSELECT	6-43
Demonstration: Polynomial Regression with PROC GLMSELECT	6-48
6.2 Generalized Linear Models.....	6-52
Demonstration: Logistic Regression	6-59
Demonstration: GENMOD Procedure.....	6-69
6.3 Mixed Models	6-78
Demonstration: Two-Way Mixed Model	6-84
6.4 Other Procedures	6-90
Exercises	6-99
6.5 Solutions	6-116
Solutions to Exercises	6-116
Solutions to Student Activities (Polls/Quizzes)	6-154

6.1 Linear Models

Objectives

- Use PROC REG to create regression and polynomial models.
- Use PROC GLM to create ANOVA and ANCOVA models.
- Use PROC GLMSELECT to perform effect selection.

3

Motivation

Use a PROC step to perform the following tasks:

- create linear models
- create statistical graphics
- save important model information

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k + \varepsilon$$



4

Duplicating the R Script

```
#Regression  
mylm = lm(salePrice ~ Gr_Liv_Area + Age_Sold)  
mylm; summary(mylm); anova(mylm)  
par(mfrow=c(2,2)); plot(mylm)  
  
#Polynomial  
x2 = x^2; x3 = x^3; x4 = x^4; x5 = x^5  
mylm = lm(y ~ x + x2 + x3 + x4 + x5)  
mylm; summary(mylm); anova(mylm)  
par(mfrow=c(2,2)); plot(mylm)  
  
#ANOVA  
mylm = lm(salePrice ~ as.factor(Heating_QC))  
mylm; anova(mylm); summary(mylm)  
par(mfrow=c(2,2)); plot(mylm)  
  
#ANCOVA  
mylm = lm(salePrice ~ as.factor(Heating_QC) + Gr_Liv_Area  
+ as.factor(Heating_QC)*Gr_Liv_Area)  
mylm; anova(mylm); summary(mylm)  
par(mfrow=c(2,2))  
plot(mylm)
```

5

Idea Exchange

What type of base R statistical functions do you use regularly?



6

REG Procedure

Use PROC REG to create a regression model.

```
mylm = lm(SalePrice ~ Gr_Liv_Area)
summary(mylm); anova(mylm)
plot(mylm)
```

```
proc reg data=ameshousing;
  model saleprice = gr_liv_area;
run;quit;
```

```
PROC REG DATA=data-set-name;
  MODEL dependent-variable = regressors </ options>;
RUN;QUIT;
```

7

Selected PROC REG statement:

MODEL specifies the dependent variables followed by an equal sign and the regressor variables.

Variables specified in the MODEL statement must be numeric variables.

REG Procedure

View the default PROC REG output.

```
mylm = lm(SalePrice ~ Gr_Liv_Area)
summary(mylm); anova(mylm)
plot(mylm)
```

```
proc reg data=ameshousing;
  model saleprice = gr_liv_area;
run;quit;
```

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	1.790671E11	1.790671E11	218.56	<.0001
Error	298	2.441564E11	819316790		
Corrected Total	299	4.232235E11			

Root MSE	28624	R-Square	0.4231
Dependent Mean	137525	Adj R-Sq	0.4212
Coeff Var	20.81348		

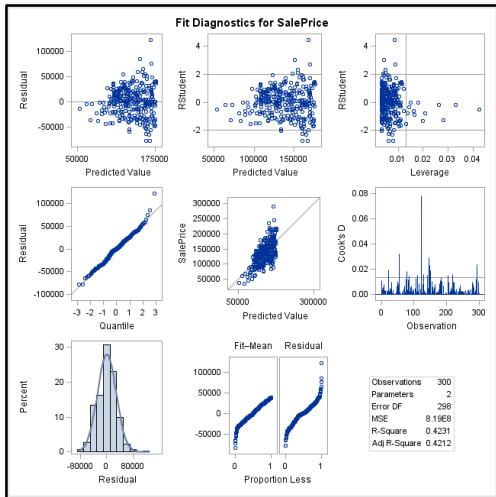
Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	18583	8213.43837	2.26	0.0244
Gr_Liv_Area	1	105.18902	7.11522	14.78	<.0001

8

PROC REG automatically produces the same tables provided by the summary() and anova() function in R. Namely, it provides the analysis of variance table and the parameter estimates table.

REG Procedure

View the default PROC REG output.



9

continued...

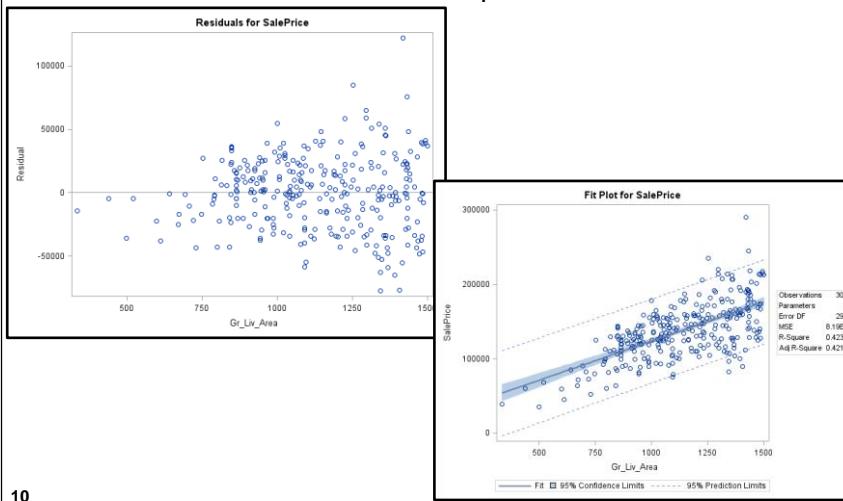
The default graphical output is similar to plotting the model object in R. The plots produced in R are the residuals versus fitted plot, the normal Q-Q plot, the standardized residuals plot, and the residuals versus leverage plot. SAS, by default, also provides, a histogram of the residuals, Cooke's D plot, the RStudent versus leverage plot, RStudent versus predicted value plot, and the response versus predicted value plot. All plots are included in a diagnostic panel.



- Use the PLOTS(UNPACK) option in the PROC REG statement to plot the default output individually, without a panel.

REG Procedure

View the default PROC REG output.



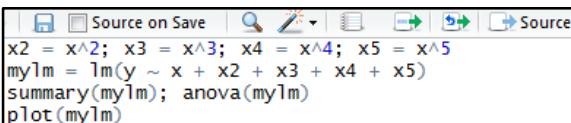
10

The plot displaying the 95% confidence and prediction bands is displayed for simple linear regression models only. In addition, a residual by predictor plot is created for each predictor specified in the model, regardless of the number of predictors.

REG Procedure

The preparation for polynomial regression is similar to R.

- Create new variables within a DATA step before you create a model within the PROC step.

```

x2 = x^2; x3 = x^3; x4 = x^4; x5 = x^5
mylm = lm(y ~ x + x2 + x3 + x4 + x5)
summary(mylm); anova(mylm)
plot(mylm)
```

```
data mydata;
  set mydata;
  x2 = x**2; x3 = x**3; x4 = x**4; x5 = x**5;
run;
```

-  Recall that SAS does not use the \wedge symbol for exponentiation.

11

PLM Procedure

To reproduce the predict() function in R, use the PLM procedure.

```

#Create Model
mylm = lm(SalePrice ~ Gr_Liv_Area)
```

Step 1: Use the STORE statement to save the model.

```
proc reg data=ameshousing;
  ...
  store mymod;
run;
```

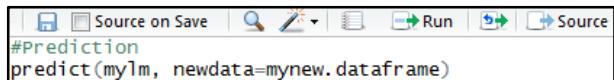
SCORE item-store-name;

12

The STORE statement requests that the procedure save the context and results of the statistical analysis. The resulting item store has a binary file format that cannot be modified. The contents of the item store can be processed with the PLM procedure.

PLM Procedure

To reproduce the predict() function in R, use the PLM procedure.



```
#Prediction
predict(myLm, newdata=mynew.dataframe)
```

Step 2: Use PROC PLM to score new data.

```
proc plm restore=mymod;
  score data=newdata out=pred;
run;
```

PROC PLM RESTORE=*item-store-specification*;
 SCORE DATA=*new-data-set*
OUT=*predicted-data-set <keywords>*;
 RUN;

13

Selected PROC PLM statements and options:

- | | |
|---------|--|
| RESTORE | specifies the source item store for processing. The value passed to the RESTORE option is the model stored from the previous analysis procedure. |
| SCORE | applies the contents of the source item store to compute predicted values. |
| DATA | specifies the input data set for scoring. |
| OUT | specifies the name of the output data set. |

PLM Procedure

SCORE Statement Keywords

Keyword	Description
PREDICTED	Linear predictor
STDERR	Standard Error
RESIDUAL	Residual
LCLM	Lower confidence limit
UCLM	Upper confidence limit
LCL	Lower prediction limit
UCL	Upper prediction limit

- ✍ Use the ALPHA= option for confidence and prediction intervals.

14

6.01 Multiple Answer Poll

Which statements are correct? (Select all that apply.)

- a. The PLM procedure uses the model specified by the STORE statement.
- b. PROC REG uses a CLASS statement to specify categorical variables.
- c. The PLM procedure SCORE statement keywords provide interval output.
- d. The PLM procedure scores new data sets.

15

STORE and SCORE Statements

- The STORE statement is supported by many SAS/STAT procedures.
 - REG, GLM, GLMSELECT, LOGISTIC, GENMOD, MIXED
 - GLIMMIX, LIFEREG, ORTHOREG, PHREG, PROBIT, SURVEYLOGISTIC, SURVEYPHREG, SURVEYREG
- The SCORE statement is used to bypass the need for the PLM procedure. The SCORE statement enables scoring new data inside the modeling procedure.
- The SCORE statement is supported only by the GLMSELECT and LOGISTIC procedures.

17



Multiple Linear Regression

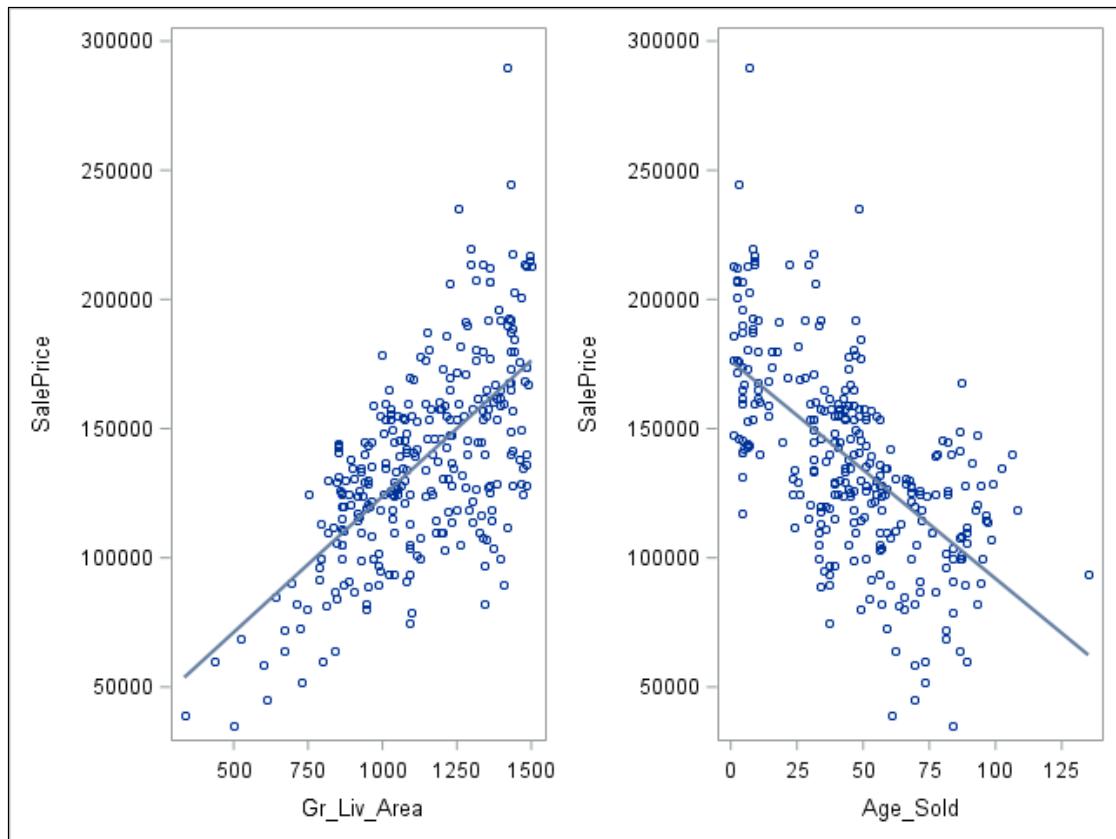
SP4R06d01.sas

1. Use PROC SGSCATTER to create a panel of the scatter plots of **SalePrice** by both **Gr_Liv_Area** and **Age_Sold** with a regression line for each.



The method of ignoring the PROC REG diagnostics plots and using PROC UNIVARIATE to analyze the residuals is simply an alternative.

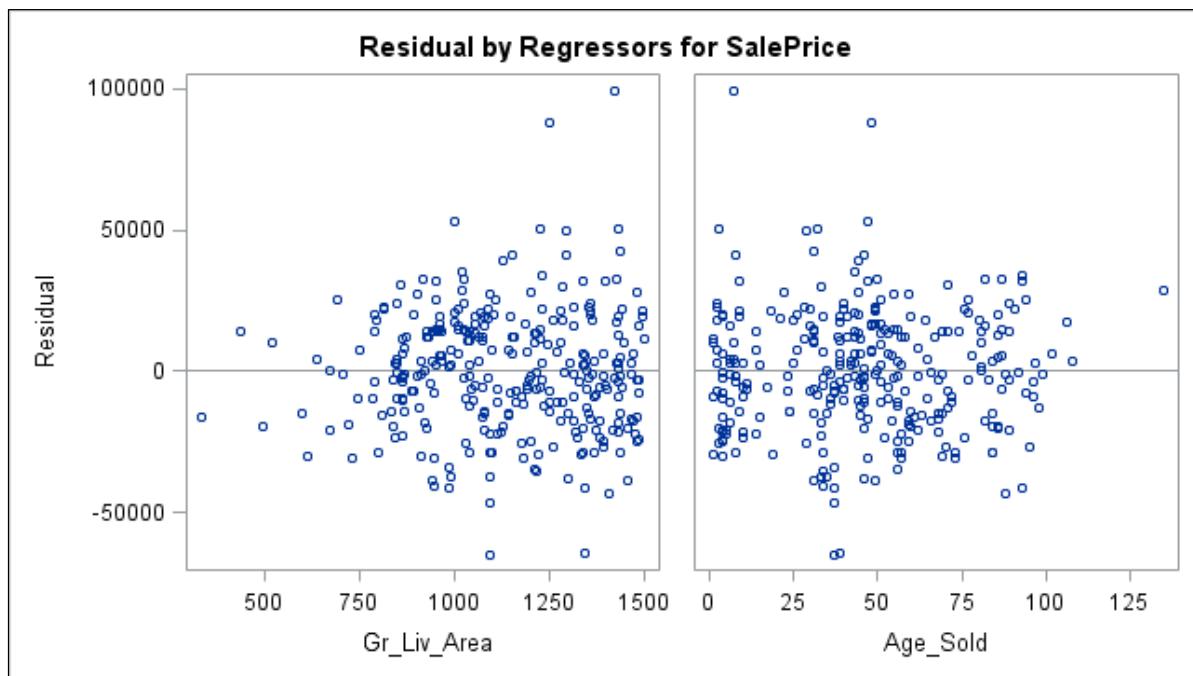
```
proc sgscatter data=sp4r.ameshousing;
  plot saleprice * (gr_liv_area age_sold) / reg;
run;
```



2. Use PROC REG to regress **SalePrice** on the variables **Gr_Liv_Area** and **Age_Sold**. Use the ODS SELECT statement to request the ANOVA, FitStatistics, ParameterEstimates table, and the plot ResidualPlot. Use the OUTPUT statement to create a new data table with the model's residuals and predicted values. Use the new data table to create a predicted versus residual plot and add a horizontal reference line at zero. Then use PROC UNIVARIATE to print the BasicMeasures table, a Q-Q plot, and a histogram with both a normal and kernel density estimate.

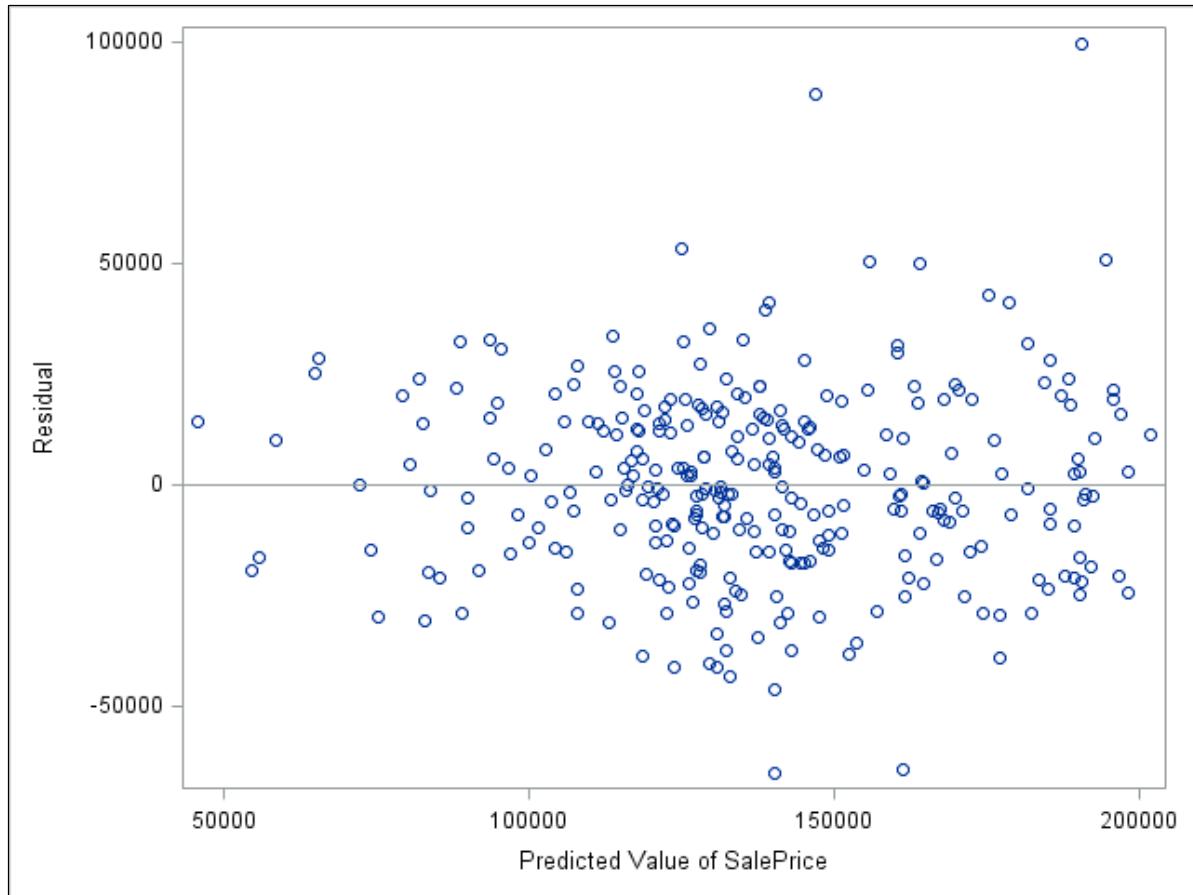
```
ods select anova fitstatistics parameterestimates residualplot;
proc reg data=sp4r.ameshousing;
  model saleprice = gr_liv_area age_sold;
  output out=sp4r.out predicted=pred residual=res rstudent=rstudent;
run;quit;
```

The REG Procedure					
Model: MODEL1					
Dependent Variable: SalePrice					
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	2.846315E11	1.423158E11	304.98	<.0001
Error	297	1.38592E11	466639714		
Corrected Total	299	4.232235E11			
Root MSE		21602	R-Square	0.6725	
Dependent Mean		137525	Adj R-Sq	0.6703	
Coeff Var		15.70759			
Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	68498	7031.03707	9.74	<.0001
Gr_Liv_Area	1	89.32657	5.47233	16.32	<.0001
Age_Sold	1	-696.90431	46.33458	-15.04	<.0001



3. Use the new data table to create a predicted versus residual plot and add a horizontal reference line at zero.

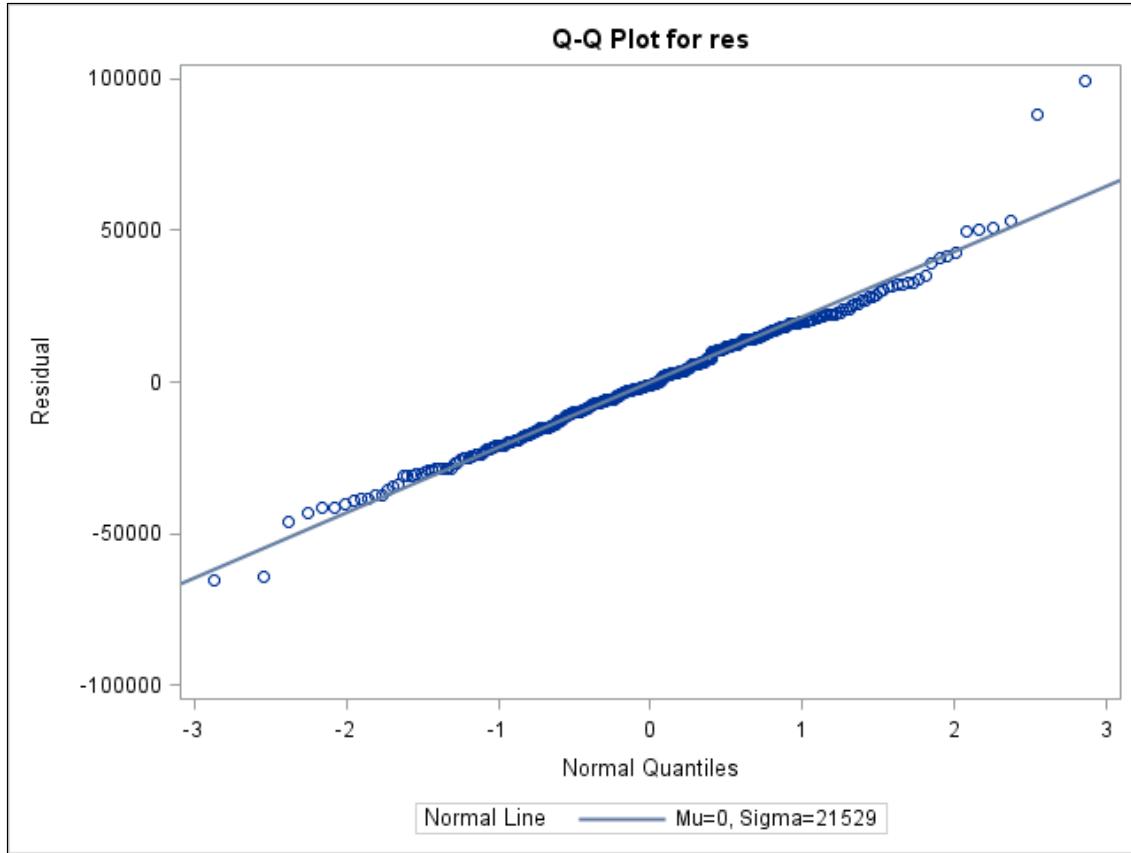
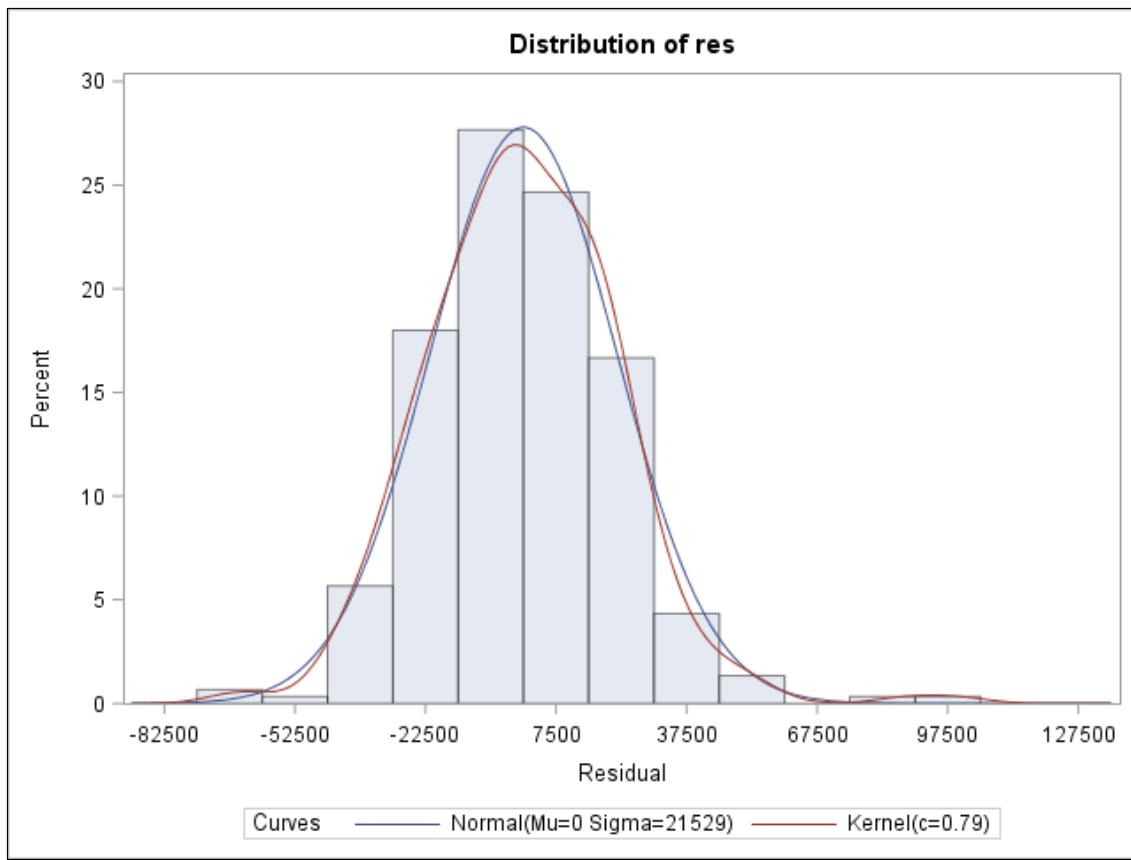
```
proc sgplot data=sp4r.out;
  scatter x=pred y=res;
  refline 0 / axis=y;
run;
```



4. Use PROC UNIVARIATE to print the BasicMeasures table, a Q-Q plot, and a histogram with both a normal and kernel density estimate.

```
ods select basicmeasures histogram qqplot;
proc univariate data=sp4r.out;
  var res;
  histogram res / normal kernel;
  qqplot res / normal(mu=est sigma=est);
run;
```

Basic Statistical Measures			
Location		Variability	
Mean	0.00	Std Deviation	21529
Median	-1029.33	Variance	463518378
Mode	.	Range	164883
		Interquartile Range	29009



5. The data set **NewData_Ames_Reg** contains five new observations from the Ames housing market. Rerun the REG procedure, but this time save the model with a STORE statement. Then, use PROC PLM to score the new data set. Request only the predicted values.

```
proc reg data=sp4r.ameshousing;
  model saleprice = gr_liv_area age_sold;
  store mymod;
run;quit;

proc plm restore=mymod;
  score data=sp4r.newdata_ames_reg out=sp4r.pred_newdata predicted;
run;
```

The PROC REG output was removed.

The PLM Procedure

Store Information

Item Store	WORK.MYMOD
Data Set Created From	WORK.AMESHOUSING
Created By	PROC REG
Date Created	29NOV15:11:27:07
Response Variable	SalePrice
Model Effects	Intercept Gr_Liv_Area Age_Sold

The PLM procedure indicates that the response variable is **SalePrice**, and the model effects **Intercept**, **Gr_Liv_Area**, and **Age_Sold** are saved.

6. Print the predictor variables and the predicted values from the scored data set.

```
proc print data=sp4r.pred_newdata;
  var saleprice gr_liv_area age_sold predicted;
run;
```

Obs	SalePrice	Gr_Liv_Area	Age_Sold	Predicted
1	213500	1338	9	181745.17
2	191500	1280	18	170292.09
3	115000	864	39	118497.25
4	160000	1145	4	167989.66
5	180000	1430	10	189266.31

End of Demonstration



Polynomial Regression

SP4R06d02.sas

Example: A researcher is interested in studying the effect of a chemical additive on paper strength. Data are collected and stored in the **Paper** data set. The independent variable of interest is the amount of chemical additive (**Amount**), and the dependent variable is the amount of force required to break the paper (**Strength**). Conduct polynomial regression with the **Paper** data set.

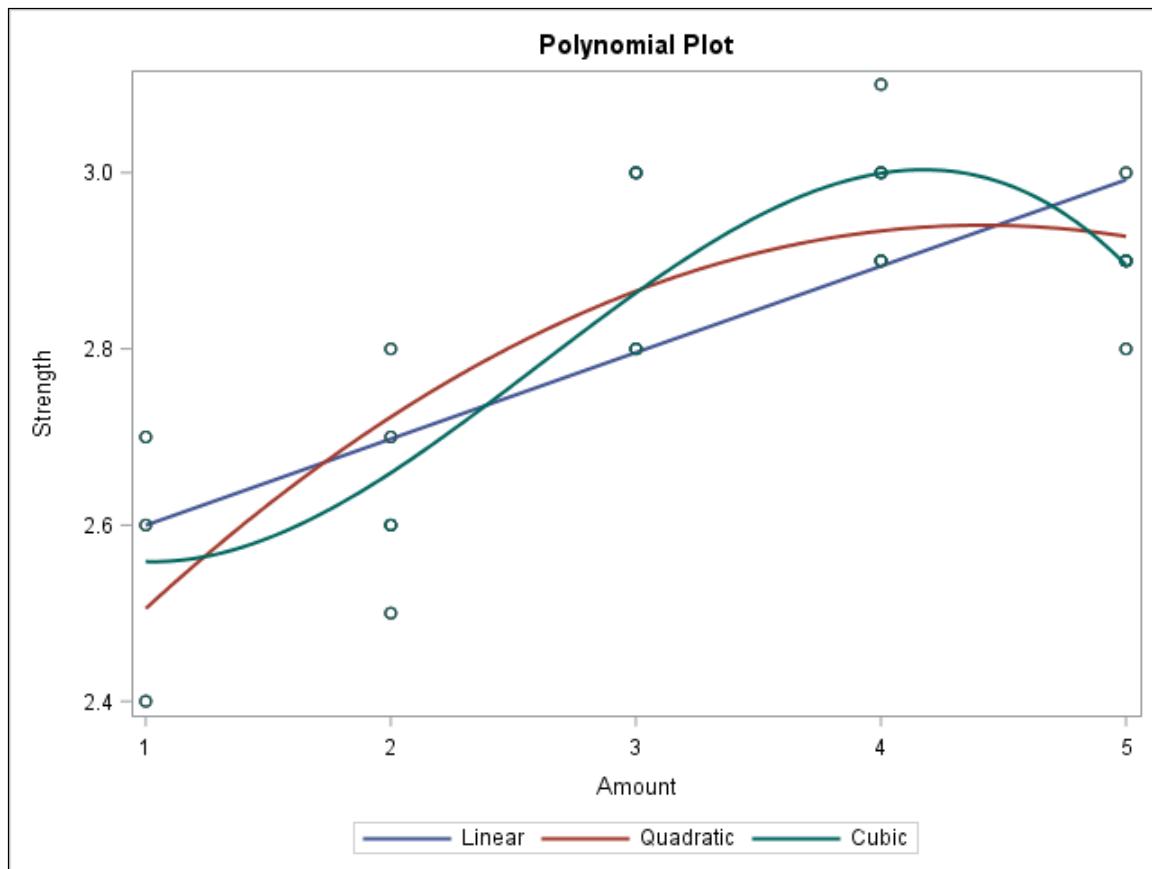
1. Print the data to view it.

```
proc print data=sp4r.paper;
run;
```

Obs	Amount	Strength
1	1	2.4
2	1	2.6
3	1	2.7
4	2	2.5
5	2	2.6
6	2	2.6
7	2	2.7
8	2	2.8
9	3	2.8
10	3	2.8
11	3	3.0
12	3	3.0
13	4	3.0
14	4	2.9
15	4	2.9
16	4	3.0
17	4	3.1
18	5	2.9
19	5	2.9
20	5	3.0
21	5	2.9
22	5	2.8

2. Use PROC SGPLOT to plot the data with linear, quadratic, and cubic lines. Use the DEGREE= option.

```
proc sgplot data=sp4r.paper;
  reg x=amount y=strength / legendlabel="Linear";
  reg x=amount y=strength / degree=2 legendlabel="Quadratic";
  reg x=amount y=strength / degree=3 legendlabel="Cubic";
  title 'Polynomial Plot';
run;
title;
```



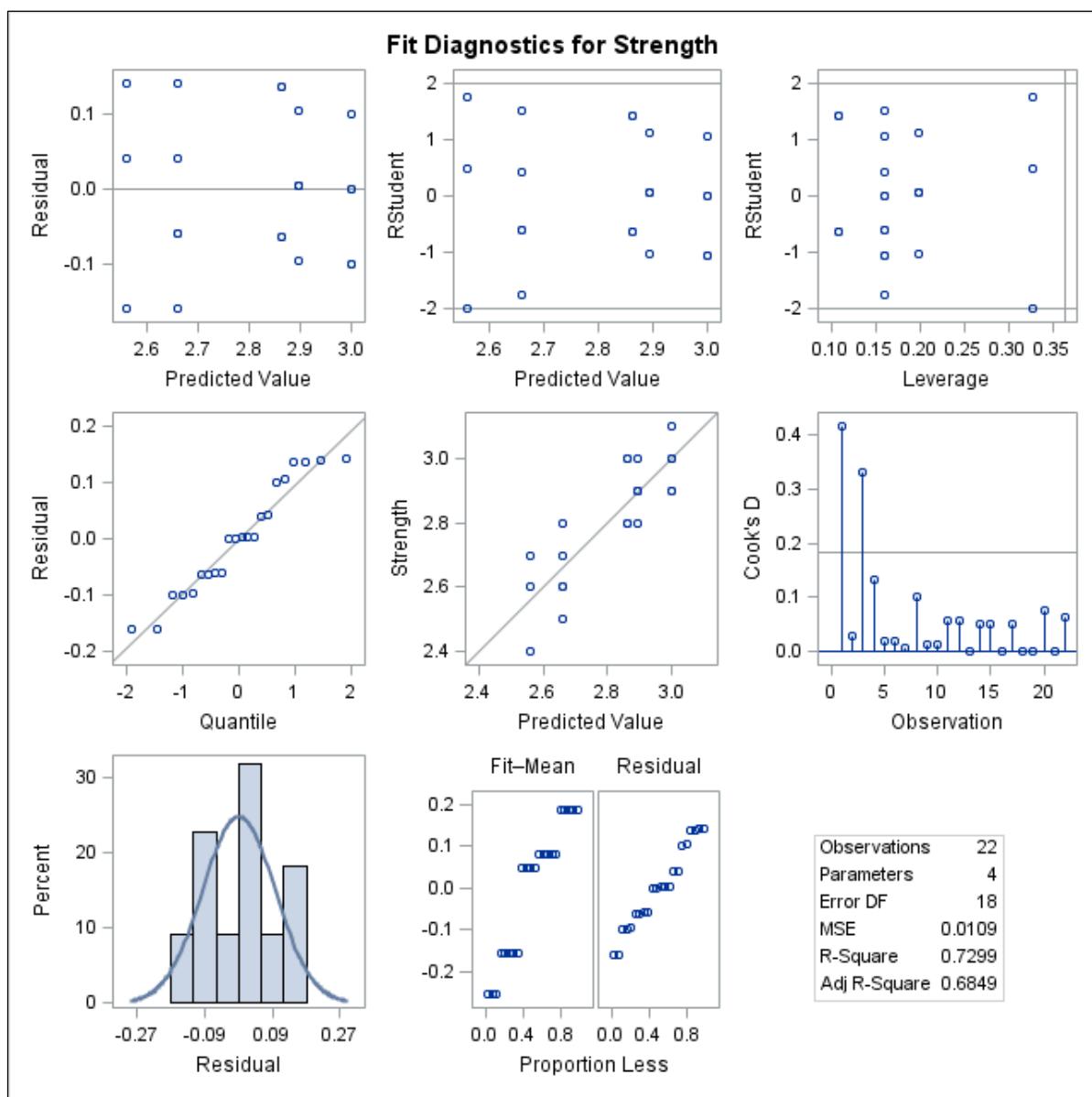
3. To the data set, add variables that represent the quadratic amount and cubic amount of chemical additive. Use PROC REG to estimate a polynomial regression model up to a cubic term.

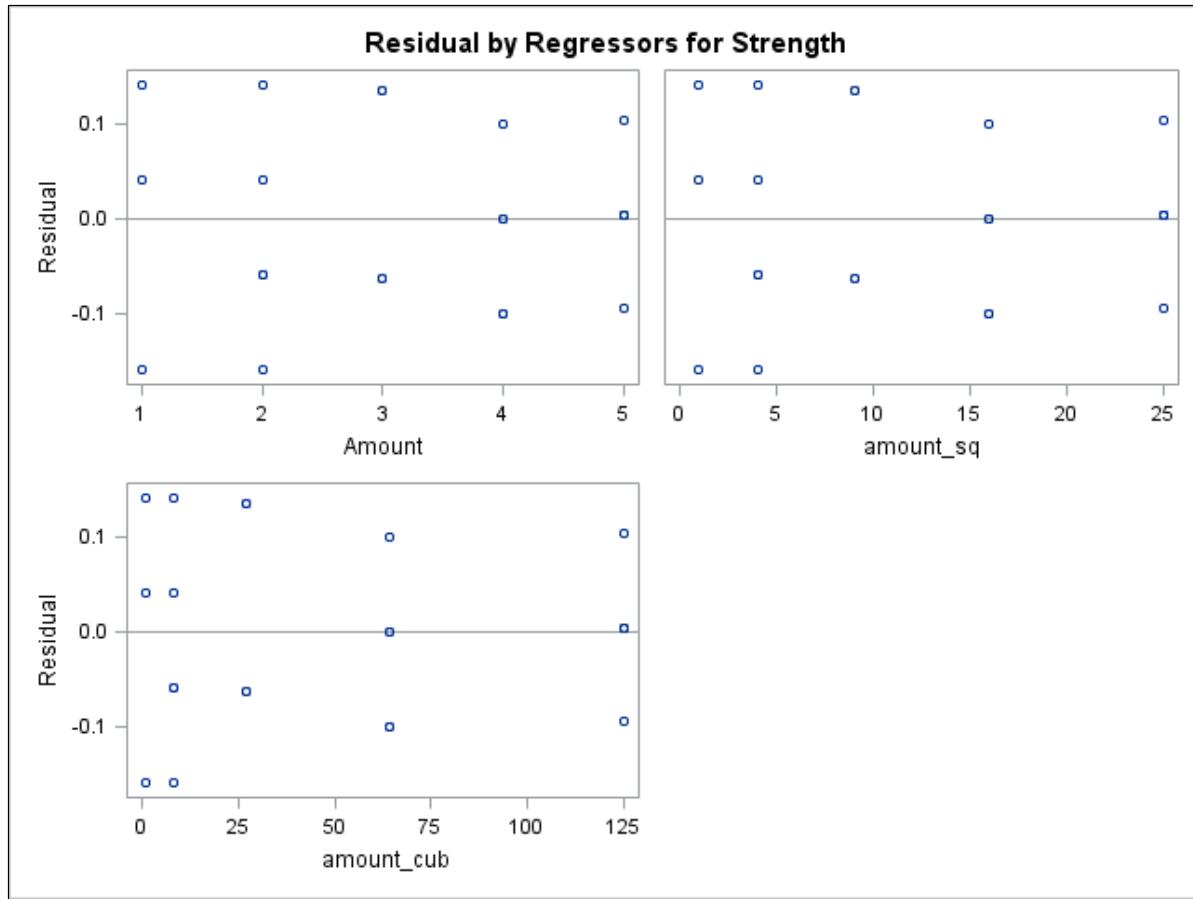
```
data sp4r.paper;
  set sp4r.paper;
  amount_sq = amount*amount;
  amount_cub = amount*amount*amount;
run;

proc reg data=sp4r.paper;
  model strength = amount amount_sq amount_cub;
run;quit;
```

The REG Procedure					
Model: MODEL1					
Dependent Variable: Strength					
Number of Observations Read					22
Number of Observations Used					22
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	0.52986	0.17662	16.22	<.0001
Error	18	0.19605	0.01089		
Corrected Total	21	0.72591			

Root MSE	0.10436	R-Square	0.7299
Dependent Mean	2.81364	Adj R-Sq	0.6849
Coeff Var	3.70919		
Parameter Estimates			
Variable	DF	Parameter Estimate	Standard Error
Intercept	1	2.73280	0.26060
Amount	1	-0.36900	0.32208
amount_sq	1	0.22339	0.11651
amount_cub	1	-0.02862	0.01270





End of Demonstration

Motivation

Use a PROC GLM to do the following tasks:

- perform an ANOVA

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

- perform an ANCOVA

$$Y_{ij} = \mu + \alpha_i + \beta_1 X_{ij} + \theta_i X_{ij} + \varepsilon_{ij}$$



20

GLM Procedure

Use PROC GLM to create an ANOVA or ANCOVA model.

```
Source on Save | Source | Run | Source
mylm = lm(SalePrice ~ as.factor(Heating_QC))
anova(mylm); summary(mylm)
```

```
proc glm data=ameshousing;
  class heating_qc (ref='Fa');
  model saleprice = heating_qc / solution;
run;quit;
```

```
PROC GLM DATA=data-table-name;
  CLASS variables <options>;
  MODEL dependent-variable = independent-variables
    </options>;
RUN;
```

21

Selected PROC GLM statements and options:

CLASS specifies the classification or categorical variables to be used in the model. Notice that the CLASS statement indicates only that the variable is a classification variable. The variable must be added to the MODEL statement to be included as a predictor.

The CLASS statement in PROC GLM creates columns in the design matrix for each classification variable. The number of columns is the same as the number of levels in the CLASS variable. The value of each design column is either 0 or 1 across all observations.

REF specifies the class level to be used as the reference level for reference coding.

MODEL specifies the dependent variable followed by an equal sign and the independent effects.
 SOLUTION requests the parameter estimates table.

GLM Procedure

View the default PROC GLM output.

```
Source on Save | Run | Source
mylm = lm(salePrice ~ as.factor(Heating_QC))
anova(mylm); summary(mylm)
```

```
proc glm data=ameshousing;
  class heating_qc (ref='Fa');
  model saleprice = heating_qc / solution;
run;quit;
```

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	66835556221	22278518740	18.50	<.0001
Error	296	356387963289	1204013389.5		
Corrected Total	299	423223519511			

R-Square	Coeff Var	Root MSE	SalePrice Mean
0.157920	25.23100	34698.90	137524.9

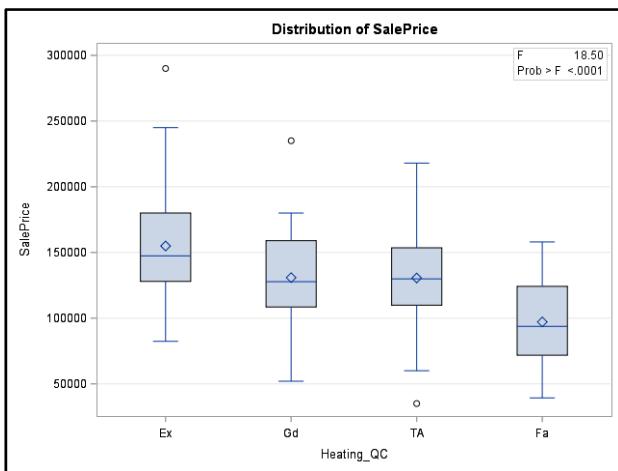
Source	DF	Type I SS	Mean Square	F Value	Pr > F
Heating_QC	3	66835556221	22278518740	18.50	<.0001

Source	DF	Type III SS	Mean Square	F Value	Pr > F
Heating_QC	3	66835556221	22278518740	18.50	<.0001

22

GLM Procedure

View the default PROC GLM output.



23

GLM Procedure

Use the SOLUTION option in the MODEL statement to request a table of parameter estimates and standard errors.

```
proc glm data=ameshousing;
  class heating_qc (ref='Fa');
  model saleprice = heating_qc / solution;
run;quit;
```

Parameter	Estimate	Standard Error	t Value	Pr > t
Intercept	97118.75000	B	8674.724021	11.20 <.0001
Heating_QC Ex	57800.43692	B	9300.714942	6.21 <.0001
Heating_QC Gd	33725.33621	B	9798.453367	3.44 0.0007
Heating_QC TA	33454.77941	B	9239.512780	3.62 0.0003
Heating_QC Fa	0.00000	B	.	.

24

Terms whose estimates are followed by the letter ‘B’ are not uniquely estimable.

GLM Procedure

Use the MEANS statement and the HOVTEST option to request a test for the assumption of equal variances.

```
proc glm data=ameshousing;
  ...
  means heating_qc / hovtest=bf;
run;quit;
```

MEANS class-variable </ HOVTEST=test-name>;

Level of Heating_QC	N	SalePrice	
		Mean	Std Dev
Ex	107	154919.187	36822.8795
Gd	58	130844.086	34912.5027
TA	119	130573.529	32177.4508
Fa	16	97118.750	37423.5437

HOVTEST=	Homogeneity of Variance Test
BARTLETT	Bartlett's Test
BF	Brown and Forsythe's Test
LEVENE	Levene's Test
OBRIEN	O'Brien's Test

25

Selected PROG GLM statements and options:

MEANS computes the arithmetic means and standard deviations of the response variables across the levels of the specified categorical variables.

HOVTEST requests a homogeneity of variance test for the groups defined by the MEANS statement.

 The BF option specifies Brown and Forsythe's variation of Levene's test.

GLM Procedure

Use the LSMEANS statement and the ADJUST= option to request multiple simultaneous comparisons.

```
proc glm data=ameshousing;
  ...
  lsmeans heating_qc / adjust=tukey;
run;quit;
```

LSMEANS class-variable </ options>;

ADJUST=	Homogeneity of Variance Test
TUKEY	Tukey Adjustment
BON	Bonferroni Adjustment
DUNNET	Dunnett Adjustment
SCHEFFE	Scheffé Adjustment

26

Selected PROG GLM statements and options:

LSMEANS computes least squares means for each classification effect that is specified.

ADJUST requests multiple comparison adjustment for the *p*-values and confidence limits for the differences of LS means.

GLM Procedure

View the default LSMEANS output.

```
proc glm data=ameshousing;
  ...
  lsmeans heating_qc / adjust=tukey;
run;quit;
```

LSMEANS class-variable </ options>;

Heating_QC	SalePrice LSMEAN	LSMEAN Number
Ex	154919.187	1
Gd	130844.086	2
TA	130573.529	3
Fa	97118.750	4

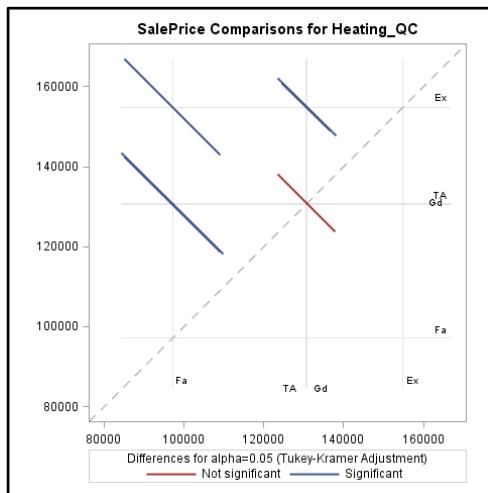
Least Squares Means for effect Heating_QC Pr > t for H0: LSMean(i)=LSMean(j) Dependent Variable: SalePrice					
i/j	1	2	3	4	
1		0.0002	<.0001	<.0001	
2	0.0002		1.0000	0.0037	
3	<.0001	1.0000		0.0020	
4	<.0001	0.0037	0.0020		

27

GLM Procedure

View the default LSMEANS output.

28



The default LSMEANS statement output is a graphical display of significant comparison. Comparisons in blue are significantly different. Lines that intersect the reference line indicate non-significance.

6.02 Multiple Choice Poll

Choose the correct statements.

- The CLASS statement is equivalent to the as.factor() function in R.
- The MEANS statement provides the same default output as PROC MEANS.
- The ADJUST= option in the LSMEANS statement is used to adjust for multiple comparisons.
- The SOLUTION option in the MODEL statement is used to display parameter estimates.

29

GLM Procedure

The ESTIMATE statement enables you to test linear functions of the parameters.

$$\hat{L}\hat{\beta}$$

- Use the ESTIMATE statement to specify the L matrix.

31

In R, a vector of coefficients is computed to represent the L matrix. In SAS, the coefficients are specified directly in the PROC step.

GLM Procedure

Test the linear combination: $\hat{\mu}_1 - \hat{\mu}_2$

```
proc glm data=ameshousing;
  ...
  estimate 'mu1-mu2'
    heating_qc 1 -1 0 0;
run;quit;
```

ESTIMATE 'estimate-name' class-variable
linear-combination </ options>;

Parameter	Estimate	Standard Error	t Value	Pr > t
mu1-mu2	24075.1007	5657.85411	4.26	<.0001

32

Selected PROG GLM statement:

ESTIMATE enables you to estimate linear functions of the parameters by creating the L matrix.

The linear functions is checked for estimability. The estimate of $L\beta$, where $\beta = (X'X)^{-1}X'Y$

is displayed along with its associated standard error, $\sqrt{L(X'X)^{-1}L's^2}$, and t test.

GLM Procedure

Test the linear combination: $\hat{\mu}_1 - \frac{\hat{\mu}_2}{3} - \frac{\hat{\mu}_3}{3} - \frac{\hat{\mu}_4}{3}$

```
proc glm data=ameshousuing;
  ...
  estimate 'mul vs the rest'
    heating_qc 3 -1 -1 -1 / divisor=3;
run;quit;
```

ESTIMATE 'estimate-name' class-variable
linear-combination </ options>;

Parameter	Estimate	Standard Error	t Value	Pr > t
mu1 vs the rest	35407.0650	4800.45834	7.38	<.0001

33

Selected ESTIMATE statement option:

DIVISOR specifies a value by which to divide all coefficients so that fractional coefficients can be entered as integer numerators.

GLM Procedure

Use the E option in the ESTIMATE statement to print the L matrix.

```
proc glm data=ameshousuing;
  ...
  estimate 'mul vs the rest'
    heating_qc 3 -1 -1 -1 / e divisor=3;
run;quit;
```

Coefficients for Estimate mu1 vs the rest	
	Row 1
Intercept	0
Heating_QC Ex	1
Heating_QC Gd	-0.333333333
Heating_QC TA	-0.333333333
Heating_QC Fa	-0.333333333

34

Selected ESTIMATE statement option:

E displays the entire L vector.

 This option is useful when you confirm the ordering of parameters for specifying L .

GLM Procedure

Use PROC GLM to also estimate an ANCOVA model.

$$Y_{ij} = \mu + \alpha_i + \beta_1 X_{ij} + \theta_i X_{ij} + \varepsilon_{ij}$$

Use the same PROC GLM statements.

- CLASS
- MODEL
- LSMEANS
- ESTIMATE
- OUTPUT

35

The ANCOVA model can be written as shown above, where

Y_{ij} the j^{th} observed response value in the i^{th} treatment group

μ overall intercept

α_i the effect of the i^{th} treatment

β_1 overall slope

θ_i effect of the i^{th} treatment on the slope

X_{ij} the j^{th} value of the covariate in the i^{th} treatment group

ε_{ij} random error



Two-Way ANOVA

SP4R06d03.sas

1. Use the **AmesHousing** data set to create a two-way ANOVA. Use the independent variables **Heating_Qc** and **Central_Air** to model **SalePrice**. Use the LSMEANS statement to identify the model's simple means and the main effect means. Finally, use the ESTIMATE statement to estimate the main effect **EX vs GD** for the **Heating_Qc** variable.

For specifying the ESTIMATE statement coefficients:

$$Y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk}$$

$$E(\bar{Y}_{1..} - \bar{Y}_{2..}) = \alpha_1 - \alpha_2 + (\bar{\alpha}\bar{\beta})_{1..} - (\bar{\alpha}\bar{\beta})_{2..}$$

```
proc glm data=sp4r.ameshousing plots=diagnostics;
  class heating_qc (ref='Fa') central_air (ref='N');
  model saleprice = heating_qc|central_air / solution;
  *Alternative MODEL statement syntax;
  *model saleprice = heating_qc central_air heating_qc*central_air;
  lsmeans heating_qc|central_air;
  estimate 'Main Effect EX vs GD' heating_qc 1 -1 0 0
            heating_qc*central_air .5 .5 -.5 -.5 0 0 0 0 / e;
run;quit;
```

Selected LSMEANS statement options:

PLOTS controls the plots produced through the ODS Graphics.

To specify more than one plot, use the PLOTS(ONLY)= option and list the names of the desired plots.

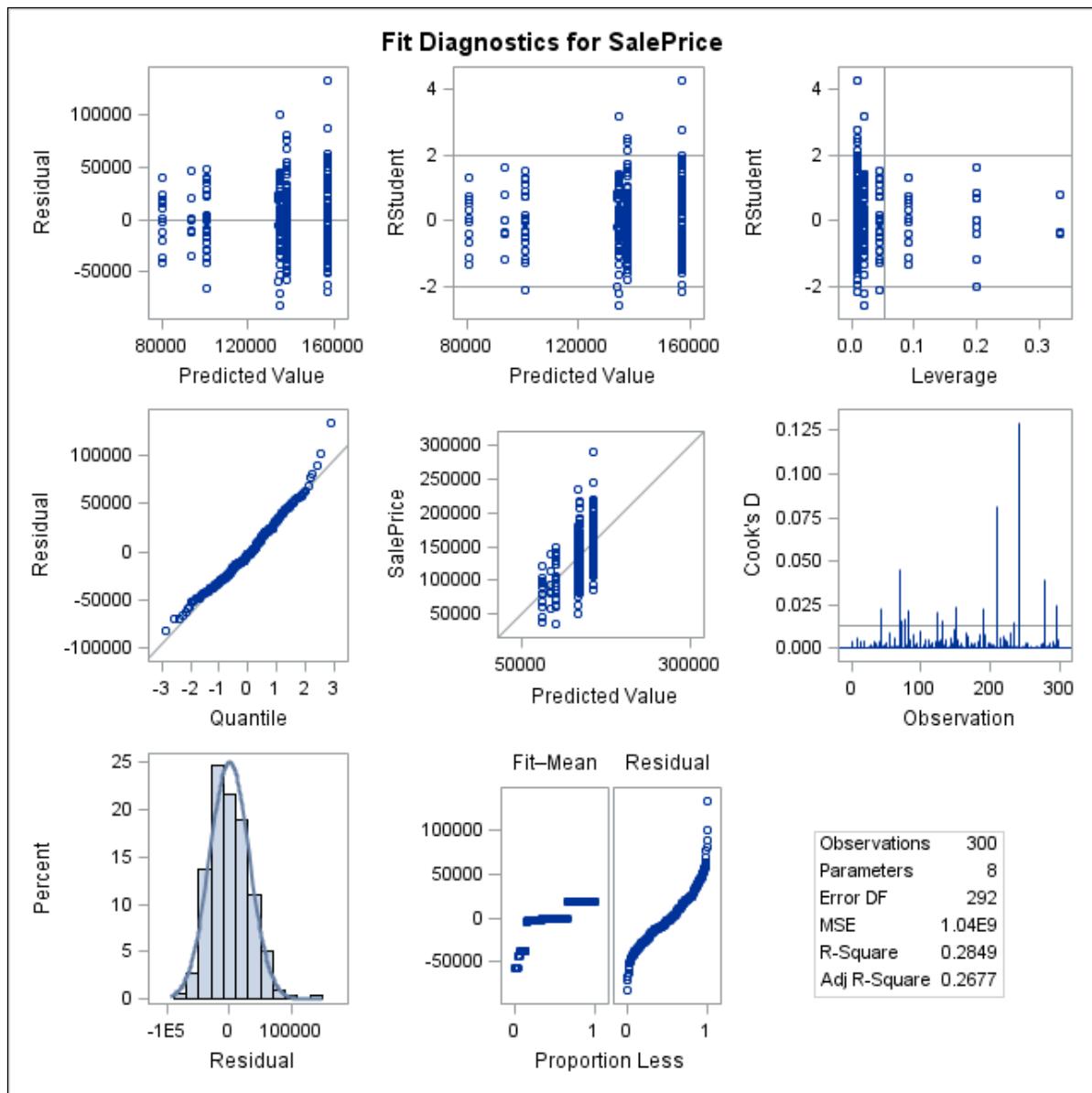
SLICE specifies effects within which to test for differences between interaction LS mean effects. This is more commonly known as tests of simple effects.

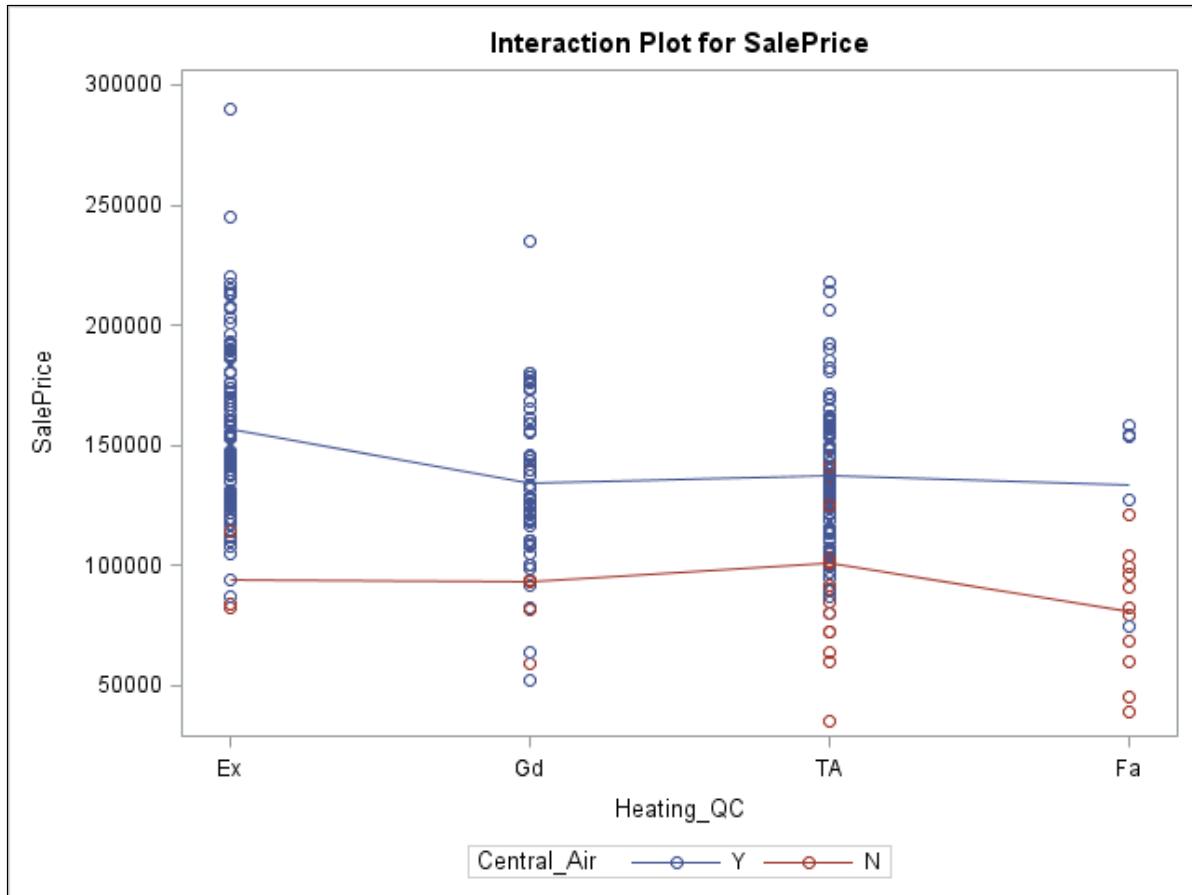
Partial Output

The GLM Procedure			
Class Level Information			
Class	Levels	Values	
Heating_QC	4	Ex Gd TA Fa	
Central_Air	2	Y N	
Number of Observations Read		300	
Number of Observations Used		300	

Dependent Variable: SalePrice					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	120567255216	17223893602	16.62	<.0001

Error	292	302656264294	1036494055.8		
Corrected Total	299	423223519511			
R-Square	Coeff Var	Root MSE	SalePrice	Mean	
0.284878	23.41004	32194.63		137524.9	
Source	DF	Type I SS	Mean Square	F Value	Pr > F
Heating_QC	3	66835556221	22278518740	21.49	<.0001
Central_Air	1	51470655579	51470655579	49.66	<.0001
Heating_Q*Central_Ai	3	2261043416	753681139	0.73	0.5365
Source	DF	Type III SS	Mean Square	F Value	Pr > F
Heating_QC	3	2659759399	886586466	0.86	0.4647
Central_Air	1	41308462327	41308462327	39.85	<.0001
Heating_Q*Central_Ai	3	2261043416	753681139	0.73	0.5365
Parameter		Estimate	Standard Error	t Value	Pr > t
Intercept		80536.36364 B	9707.04550	8.30	<.0001
Heating_QC	Ex	13089.96970 B	20969.61495	0.62	0.5330
Heating_QC	Gd	13023.63636 B	17364.49088	0.75	0.4538
Heating_QC	TA	20513.63636 B	11802.19011	1.74	0.0832
Heating_QC	Fa	0.00000 B	.	.	.
Central_Air	Y	53063.63636 B	17364.49088	3.06	0.0025
Central_Air	N	0.00000 B	.	.	.
Heating_Q*Central_Ai	Ex Y	9997.28030 B	25631.81326	0.39	0.6968
Heating_Q*Central_Ai	Ex N	0.00000 B	.	.	.
Heating_Q*Central_Ai	Gd Y	-12262.18353 B	22986.53620	-0.53	0.5941
Heating_Q*Central_Ai	Gd N	0.00000 B	.	.	.
Heating_Q*Central_Ai	TA Y	-16466.76136 B	18904.69031	-0.87	0.3844
Heating_Q*Central_Ai	TA N	0.00000 B	.	.	.
Heating_Q*Central_Ai	Fa Y	0.00000 B	.	.	.
Heating_Q*Central_Ai	Fa N	0.00000 B	.	.	.





The GLM Procedure
Least Squares Means

Heating_QC	SalePrice LSMEAN
Ex	125156.792
Gd	113960.726
TA	119348.438
Fa	107068.182

The GLM Procedure
Least Squares Means

Central_Air	SalePrice LSMEAN
Y	140573.894
N	92193.174

The GLM Procedure
Least Squares Means

Heating_QC	Central_Air	SalePrice LSMEAN
Ex	Y	156687.250
Ex	N	93626.333
Gd	Y	134361.453
Gd	N	93560.000
TA	Y	137646.875
TA	N	101050.000
Fa	Y	133600.000
Fa	N	80536.364

Coefficients for Estimate Main Effect EX vs GD

Row 1

Intercept		0
Heating_QC	Ex	1
Heating_QC	Gd	-1
Heating_QC	TA	0
Heating_QC	Fa	0
Central_Air	Y	0
Central_Air	N	0
Heating_Q*Central_Ai	Ex Y	0.5
Heating_Q*Central_Ai	Ex N	0.5
Heating_Q*Central_Ai	Gd Y	-0.5
Heating_Q*Central_Ai	Gd N	-0.5
Heating_Q*Central_Ai	TA Y	0
Heating_Q*Central_Ai	TA N	0
Heating_Q*Central_Ai	Fa Y	0
Heating_Q*Central_Ai	Fa N	0

Dependent Variable: SalePrice

Parameter	Estimate	Standard		
		Error	t Value	Pr > t
Main Effect EX vs GD	11196.0653	12065.6497	0.93	0.3542

End of Demonstration



ANCOVA

SP4R06d04.sas

1. Create an ANCOVA model with the **SalePrice** as the dependent variable. Use the categorical variable **Heating_Qc** and the continuous variable **Gr_Liv_Area** as the independent variables. Use the LSMEANS statement to create simultaneous comparisons of the **Heating_Qc** variable means and use a Tukey adjustment. Finally, use the ESTIMATE statement to predict the value for **Heating_Qc** category EX and the mean number of square feet for the ground living area, **Gr_Liv_Area**.

```
proc sql;
  select mean(gr_liv_area) into :gr_mean from sp4r.ameshousing;
quit;

proc glm data=sp4r.ameshousing plots=diagnostics;
  class heating_qc (ref='Fa');
  model saleprice = heating_qc|gr_liv_area / solution clparm;
  lsmeans heating_qc / at gr_liv_area=&gr_mean pdiff adjust=tukey cl;
  estimate 'My Estimate' intercept 1 heating_qc 1 0 0 0
    gr_liv_area &gr_mean heating_qc*gr_liv_area &gr_mean;
run;quit;
```

Selected LSMEANS statement options:

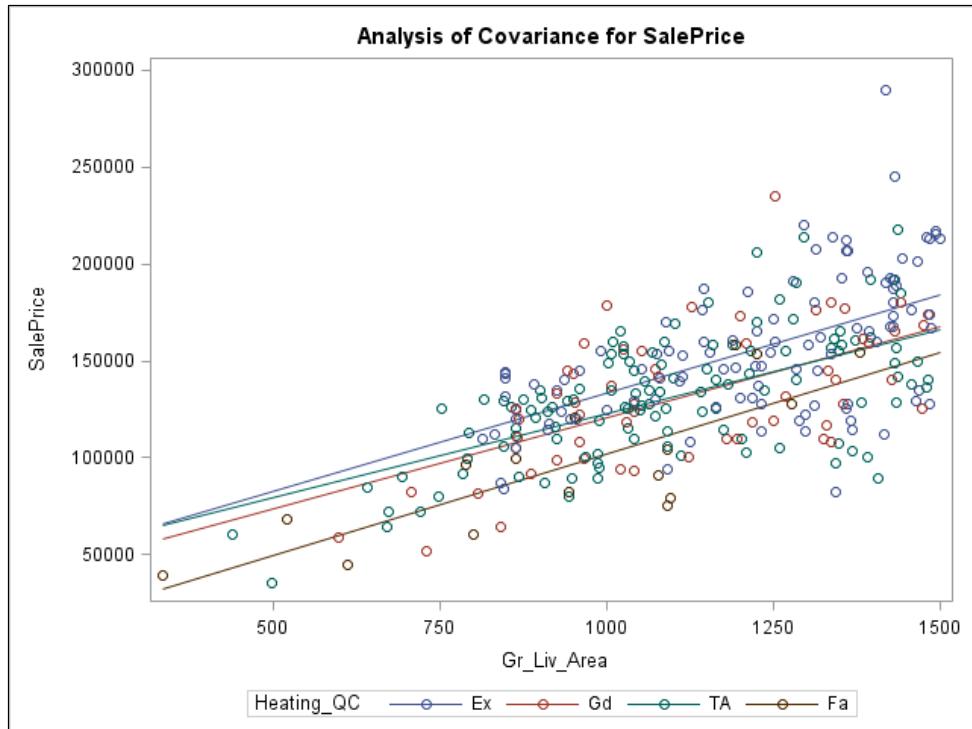
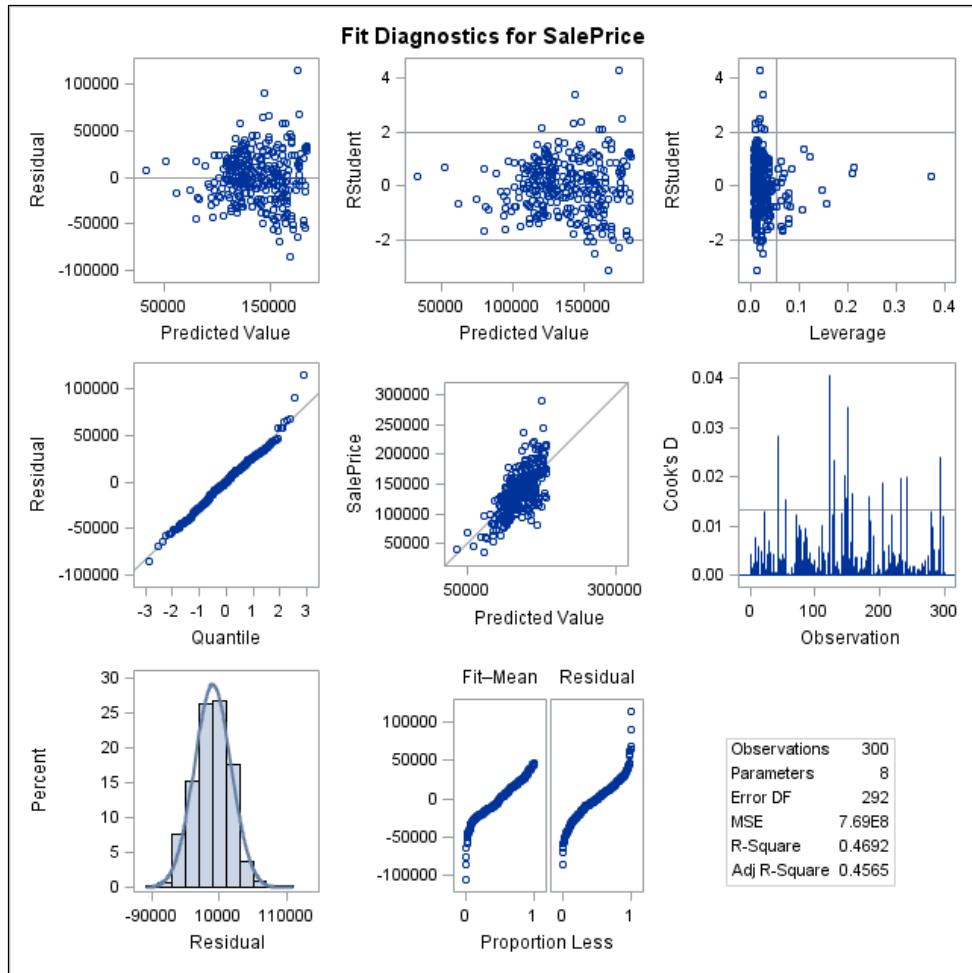
- AT enables you to modify the values of the covariates used in computing LS means.
- CLPARM produces confidence limits for the parameter estimates (if the SOLUTION option is also specified) and the results of all ESTIMATE statements.
- CL requests confidence limits for the individual LS means. If you specify the PDIFF option, confidence limits for differences between means are produced as well. You can control the confidence level with the ALPHA=option.

Partial Output

The GLM Procedure				
Class Level Information				
Class	Levels	Values		
Heating_QC	4	Ex Gd TA Fa		
Number of Observations Read			300	
Number of Observations Used			300	

Dependent Variable: SalePrice					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	198567145976	28366735139	36.87	<.0001
Error	292	224656373535	769371142.24		
Corrected Total	299	423223519511			

R-Square	Coeff Var	Root MSE	SalePrice	Mean	
0.469178	20.16911	27737.54	137524.9		
Source	DF	Type I SS	Mean Square	F Value	Pr > F
Heating_QC	3	66835556221	22278518740	28.96	<.0001
Gr_Liv_Area	1	131064547966	131064547966	170.35	<.0001
Gr_Liv_Ar*Heating_QC	3	667041788.72	222347262.91	0.29	0.8333
Source	DF	Type III SS	Mean Square	F Value	Pr > F
Heating_QC	3	1543210289	514403430	0.67	0.5719
Gr_Liv_Area	1	97663329926	97663329926	126.94	<.0001
Gr_Liv_Ar*Heating_QC	3	667041789	222347263	0.29	0.8333
Parameter	Estimate	Standard Error	t Value	Pr > t	
Intercept	-3075.90903 B	24734.90350	-0.12	0.9011	
Heating_QC Ex	34906.09602 B	29467.82243	1.18	0.2372	
Heating_QC Gd	29225.22929 B	30914.59607	0.95	0.3453	
Heating_QC TA	38654.98672 B	27747.96770	1.39	0.1647	
Heating_QC Fa	0.00000 B	.	.	.	
Parameter	95% Confidence Limits				
Intercept	-51757.20210	45605.38405			
Heating_QC Ex	-23090.15612	92902.34816			
Heating_QC Gd	-31618.44902	90068.90761			
Heating_QC TA	-15956.38252	93266.35595			
Heating_QC Fa	.	.			
Parameter	Estimate	Standard Error	t Value	Pr > t	
Gr_Liv_Area	105.13605 B	24.91395	4.22	<.0001	
Gr_Liv_Ar*Heating_QC Ex	-3.86407 B	28.09793	-0.14	0.8907	
Gr_Liv_Ar*Heating_QC Gd	-10.74900 B	29.82343	-0.36	0.7188	
Gr_Liv_Ar*Heating_QC TA	-17.90381 B	27.36056	-0.65	0.5134	
Gr_Liv_Ar*Heating_QC Fa	0.00000 B	.	.	.	
Parameter	95% Confidence Limits				
Gr_Liv_Area	56.10238	154.16973			
Gr_Liv_Ar*Heating_QC Ex	-59.16421	51.43607			
Gr_Liv_Ar*Heating_QC Gd	-69.44513	47.94713			
Gr_Liv_Ar*Heating_QC TA	-71.75270	35.94509			
Gr_Liv_Ar*Heating_QC Fa	.	.			

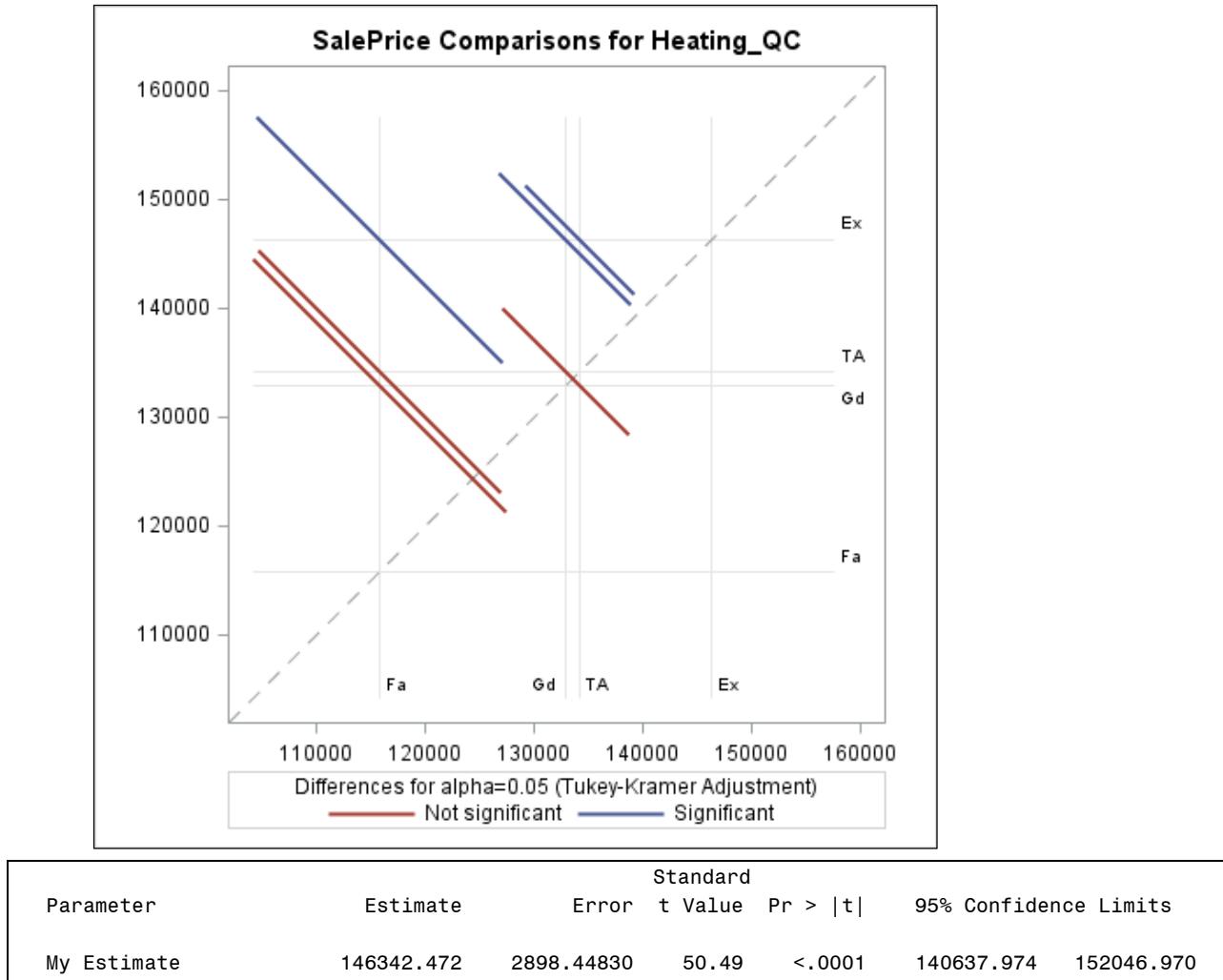


Least Squares Means at Gr_Liv_Area=1130.74 Adjustment for Multiple Comparisons: Tukey-Kramer				
	Heating_QC	SalePrice LSMEAN	LSMEAN Number	
Ex		146342.472	1	
Gd		132876.532	2	
TA		134216.070	3	
Fa		115805.632	4	

Least Squares Means for effect Heating_QC Pr > t for H0: LSMean(i)=LSMean(j)				
Dependent Variable: SalePrice				
i/j	1	2	3	4
1		0.0218	0.0106	0.0030
2	0.0218		0.9907	0.2321
3	0.0106	0.9907		0.1446
4	0.0030	0.2321	0.1446	

	Heating_QC	SalePrice LSMEAN	95% Confidence Limits	
Ex		146342	140638	152047
Gd		132877	125675	140078
TA		134216	129126	139306
Fa		115806	99613	131999

Least Squares Means for Effect Heating_QC				
		Difference Between Means	Simultaneous Confidence Limits for LSMean(i)-LSMean(j)	
i	j			
1	2	13466	1404.248457	25528
1	3	12126	2089.307907	22163
1	4	30537	7996.865820	53077
2	3	-1339.537220	-12918	10238
2	4	17071	-6196.175101	40338
3	4	18410	-3874.437835	40695



End of Demonstration

Motivation

Use stepwise model selection.

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_l X_l + \varepsilon$$



$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k + \varepsilon$$



38

Consider all possible predictors and enable the model to choose the best explanatory set.

GLMSELECT Procedure

Use PROC GLMSELECT to perform effect selection in the framework of general linear models.

General form of the GLMSELECT procedure:

```
PROC GLMSELECT DATA=data-table-name;
  CLASS categorical-variables;
  MODEL dependent-variable = model-effects / options;
RUN;
```

39

The GLMSELECT procedure combines features of PROC GLM and PROC REG for fitting general linear models, including the CLASS statement for categorical variables and automatic variable selection methods. The MODEL statement options enable you to choose the appropriate stepwise model selection criteria.

GLMSELECT Procedure

Use the SELECTION= option in the MODEL statement to specify the method used to select the model.

SELECTION=	Description
NONE	No model selection
FORWARD	Forward selection. The model starts with no effects and adds only effects.
BACKWARD	Backward selection. The model starts with all effects and deletes effects only.
STEPWISE	Stepwise regression; similar to the FORWARD method except effects in the model do not necessarily stay in the model.
LAR	Least angle regression; similar to the FORWARD method except parameter estimates are shrunk.

40



If the SELECTION= option is omitted, the default is SELECTION=STEPWISE.

GLMSELECT Procedure

Use the SELECTION= option in the MODEL statement to specify the method used to select the model.

SELECTION=	Description
LASSO	Specifies the LASSO method, which adds and deletes parameters based on a version of ordinary least squares where the sum of the absolute regression coefficients is constrained.
ELASTICNET	An extension of LASSO. Both the sum of the absolute regression coefficients and the sum of the squared regression coefficients are constrained.
GROUPLASSO	A variant of LASSO. Based on a version of ordinary least squares in which the sum of the Euclidean norms, a group of regression coefficients is constrained.

41

GLMSELECT Procedure

Use the SELECT= option in the MODEL statement to specify selection criteria.

- The SELECT= options are shown below.

ADJRSQ	AIC	AICC	BIC
CP	CV	PRESS	RSQUARE
SBC	SL	VALIDATE	

- The SELECT=SL option must be followed by either of the following options:
 - SLE= specifies significance level for entry.
 - SLS= specifies significance level for removal.

42

Selected PROC GLMSELECT MODEL statement options:

SELECT	specifies the criterion used to determine the order in which effects enter or leave (or both) at each step of the selection method.
ADJRSQ	adjusted R-square statistic
AIC	Akaike information criterion
AICC	corrected Akaike information criterion
BIC	Sawa Bayesian information criterion
CP	Mallow's C(p) statistic
CV	predicted residual sum of square with k -fold cross validation
CVEX	predicted residual sum of square with k -fold external cross validation
PRESS	predicted residual sum of squares
SBC	Schwarz Bayesian information criterion
SL	significance level
VALIDATE	average square error for the validation data.

-  The default value of the SELECT= criterion is SELECT=SBC.
-  The SELECT= option is not valid with the LAR and LASSO methods.

For details of the criteria used in model selection methods, go to *SAS/STAT® 14.1 User's Guide*, “The GLMSELECT Procedure” (http://support.sas.com/documentation/cdl/en/statug/68162/HTML/default/viewer.htm#statug_glmselect_details15.htm).

GLMSELECT Procedure

Use the CHOOSE= option in the MODEL statement to choose the model that yields the best value of the specified criterion from the selection process.

- The CHOOSE= options are shown below.

ADJRSQ	AIC	AICC	BIC	CP
CV	CVEX	PRESS	SBC	VALIDATE

- If you use CHOOSE=VALIDATE, you must specify the validation data in the PROC GLMSELECT statement (VALDATA=) or use the PARTITION statement.

43

Selected PROC GLMSELECT MODEL statement options:

CHOOSE chooses from the list of models at the steps of the selection process the model that yields the best value of the specified criterion. If the optimal value of the specified criterion occurs for models at more than one step, then the model with the smallest number of parameters is chosen.

 If no CHOOSE= option is specified then the model selected is the model at the final step in the selection process.

SELECT specifies the criterion used to determine the order in which effects enter or leave at each step of the selection method.

ADJRSQ adjusted R-square statistic

AIC Akaike information criterion

AICC corrected Akaike information criterion

BIC Sawa Bayesian information criterion

CP Mallow's C(p) statistic

CV predicted residual sum of square with k-fold cross validation

CVEX predicted residual sum of square with k-fold external cross validation

PRESS predicted residual sum of squares

SBC Schwarz Bayesian information criterion

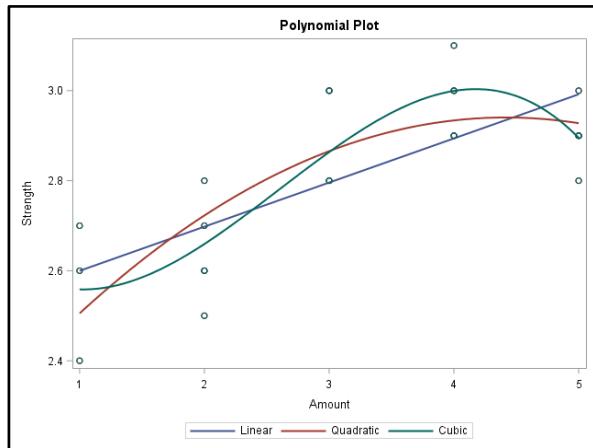
SL significance level

VALIDATE average square error for the validation data

 This option requires the user to specify a data set in the PROC GLMSELECT statement with the VALDATA= option or use the PARTITION statement to enable the procedure to split the data into training and validation data sets.

GLMSELECT Procedure

The GLMSELECT procedure can also be used to easily create polynomial predictors.



44

GLMSELECT Procedure

Use the EFFECT statement to bypass creating new variables via the DATA step for polynomial regression.

```
proc glmselect data=paper outdesign=des;
  effect x_new = polynomial(amount / degree=5);
  model strength = x_new / selection=none;
run;quit;
```

EFFECT name = effect-type(variables </ options>);

- The OUTDESIGN= option creates a new data table with the polynomial variables.

45

Here **X_New** represents the set of predictors X , X^2 , X^3 , X^4 , and X^5 . The polynomial predictors are created in the EFFECT statement and used in the MODEL statement.

GLMSELECT Procedure

The EFFECT statement creates a macro variable representing the predictors specified in the EFFECT statement.

```
proc reg data=des;
  model y = & glsmmod;
run;quit;
```

- As an alternative, the macro variable represents the final set of predictors from a model selection algorithm.

46

Notice that the data set being used here is the name from the OUTDESIGN option in the GLMSELECT procedure. Here, the & GLSMOD macro variable represents the set of predictors created in the EFFECT statement, X, X², X³, X⁴, and X⁵.

6.03 Poll

The GLMSELECT procedure can be used to create a regression model, a polynomial regression model, an ANOVA model, an ANCOVA model, and to conduct stepwise model selection.

- True
- False

47



Stepwise Selection with PROC GLMSELECT

SP4R06d05.sas

1. The **AmesHousing2** data set contains 1,361 observations from which 300 were sampled to create the **AmesHousing** data set. Use the larger data set to perform effect selection using the LASSO method in the GLMSELECT procedure. Use **SalePrice** as the dependent variable and **lot_area**, **gr_liv_area**, **garage_area**, **basement_area**, **deck_porch_area**, **age_sold**, **heating_qc**, **central_air**, **fireplaces**, and **lot_shape_2** as the independent variables. Partition the data into 50% training and 50% validation to choose the best model according to the average squared error of the validation data. Use a STORE statement to save the final model for scoring new data.

```
%let cont_vars = lot_area gr_liv_area garage_area basement_area
deck_porch_area age_sold;
%let cat_vars = heating_qc central_air fireplaces lot_shape_2;

proc glmselect data=sp4r.ameshousing2 plots=all seed=802;
  class &cat_vars;
  model saleprice = &cont_vars &cat_vars /
    selection=lasso(choose=validate stop=none);
  partition fraction(validate=0.5);
  store mymod;
run;
```

Selected PROC GLMSELECT statements and options:

SEED= specifies the seed for the random selection of training and validation data.

SELECTION= specifies the method that is used to select the model.

CHOOSE= specifies the criterion that is used for choosing the model.

STOP= specifies when PROC GLMSELECT should stop the selection process.

PARTITION specifies how observations in the input data set are logically partitioned into disjoint subsets for model training, validation, and testing.

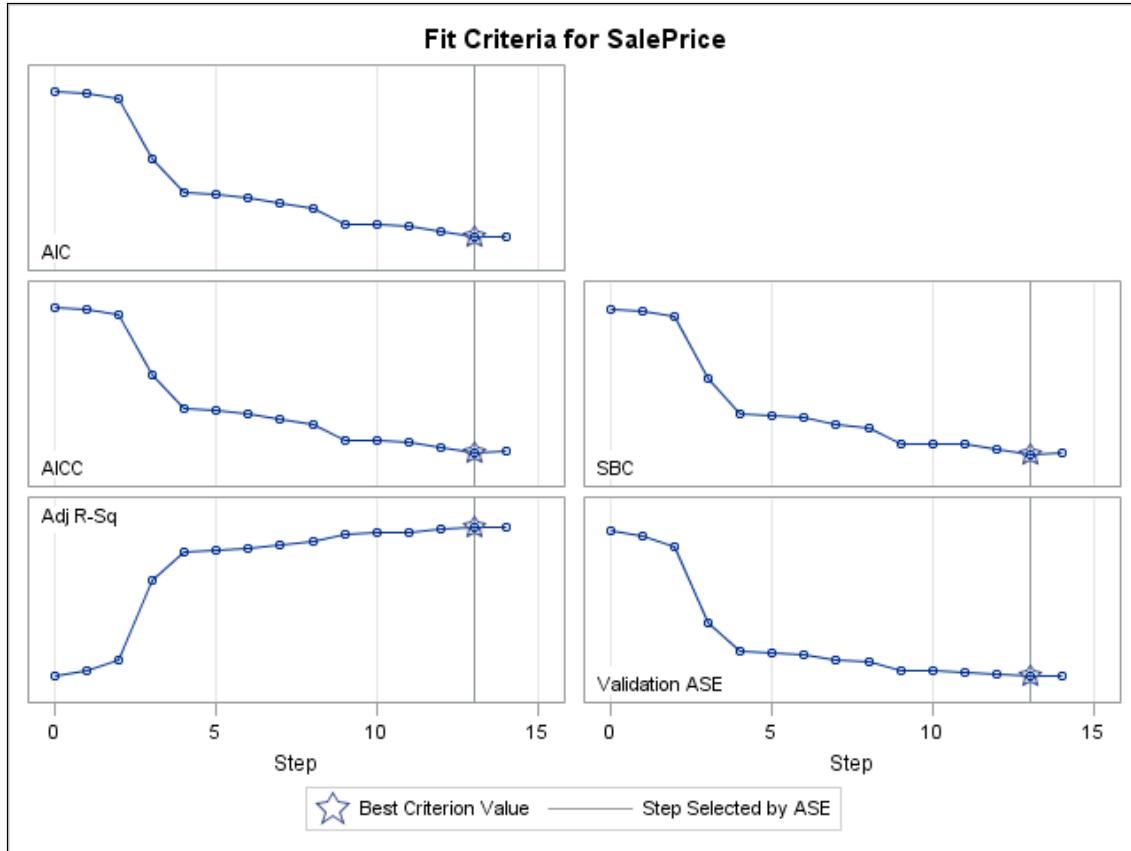
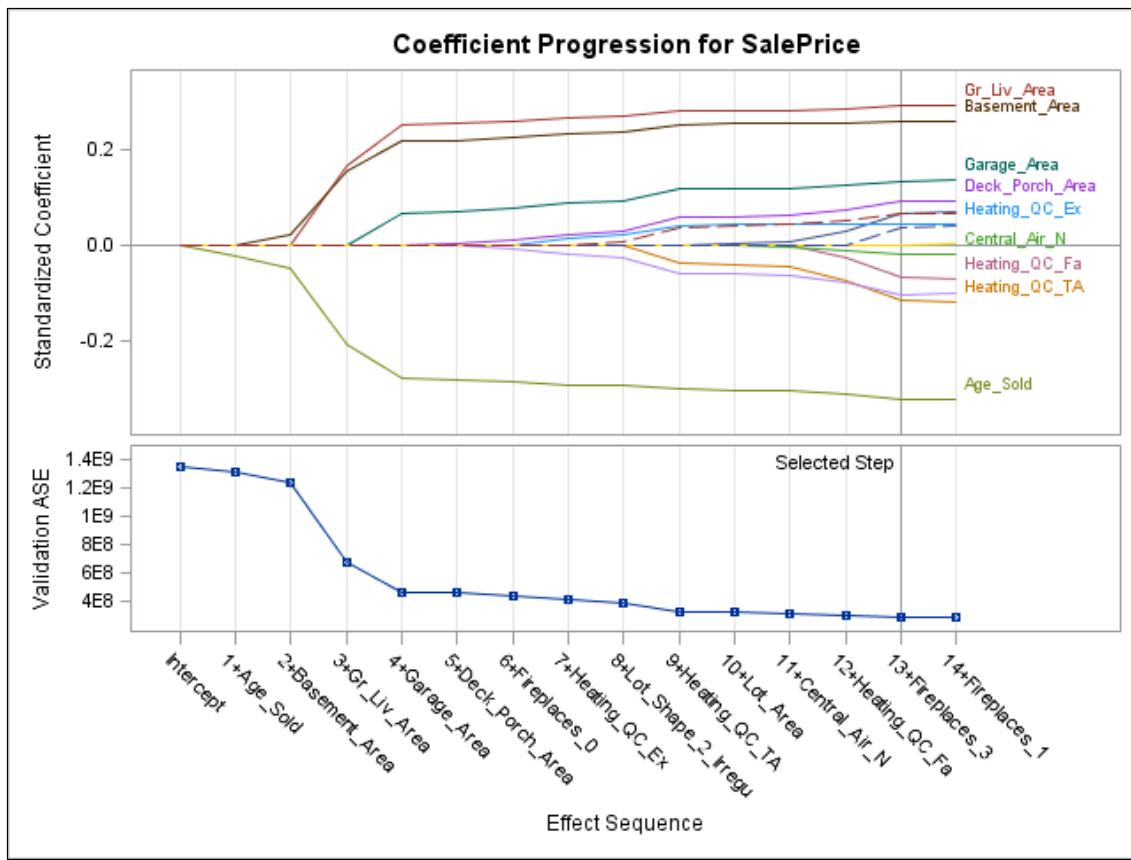
FRACTION requests that the specified proportions of the observations in the input data set be randomly assigned training and validation roles.

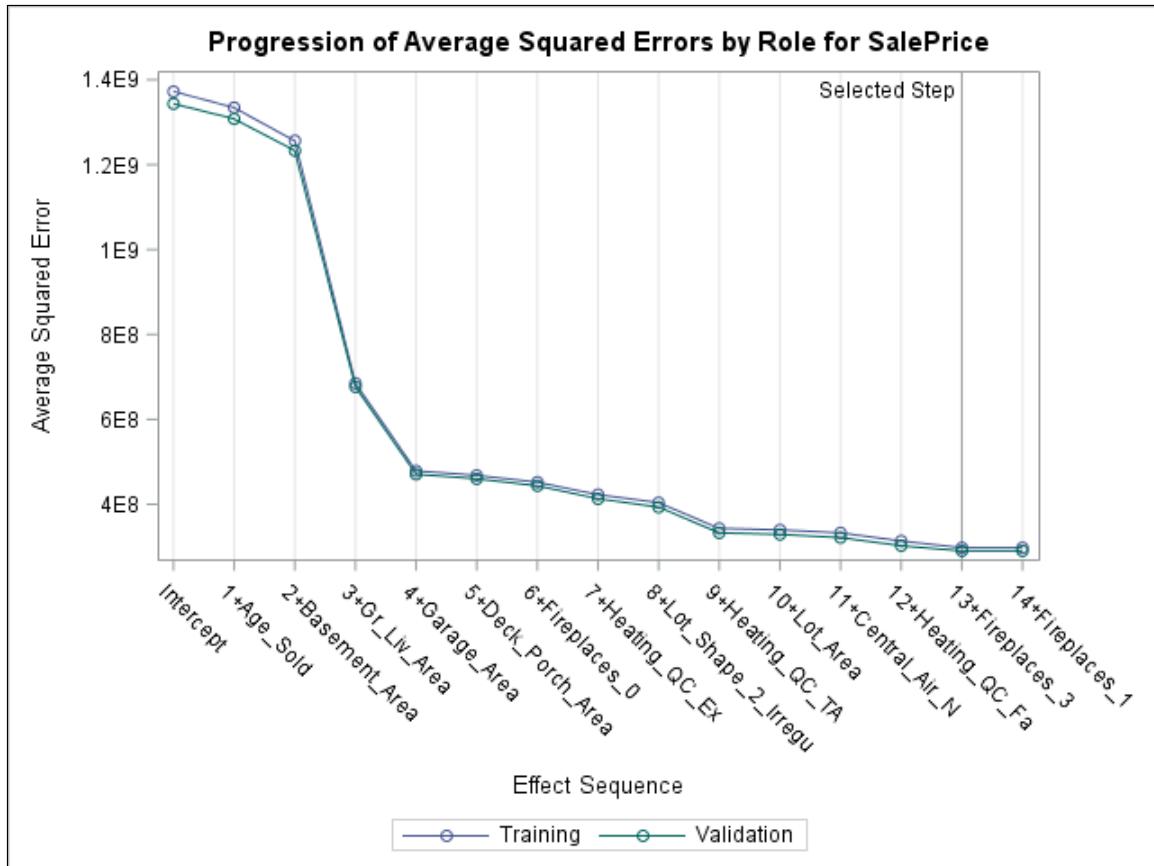
The GLMSELECT Procedure					
Data Set	SP4R.AMESHOUSING2				
Dependent Variable	SalePrice				
Selection Method	LASSO				
Stop Criterion	None				
Choose Criterion	Validation ASE				
Effect Hierarchy Enforced	None				
Random Number Seed	802				
Number of Observations Read	1361				
Number of Observations Used	1358				
Number of Observations Used for Training	674				
Number of Observations Used for Validation	684				
Class Level Information					
Class	Levels	Values			
Heating_QC	5	Ex Fa Gd Po TA			
Central_Air	2	N Y			
Fireplaces	4	0 1 2 3			
Lot_Shape_2	2	Irregular Regular			
Dimensions					
Number of Effects	11				
Number of Effects after Splits	20				
Number of Parameters	20				

LASSO Selection Summary					
Step	Effect Entered	Effect Removed	Number Effects In	ASE	Validation ASE
0	Intercept		1	1372325158	1343571005
1	Age_Sold		2	1334337999	1307767238
2	Basement_Area		3	1256197047	1232959707
3	Gr_Liv_Area		4	684824360	676605508
4	Garage_Area		5	478347523	470526968
5	Deck_Porch_Area		6	468166048	460114720
6	Fireplaces_0		7	451890629	443302771
7	Heating_QC_Ex		8	422582721	412990350
8	Lot_Shape_2_Irregular		9	403472784	393310166
9	Heating_QC_TA		10	343050976	332512247
10	Lot_Area		11	339627720	328917810
11	Central_Air_N		12	332817601	321415534
12	Heating_QC_Fa		13	313271598	302330452
13	Fireplaces_3		14	297520664	290198671*
14	Fireplaces_1		15	297453155	290347744

* Optimal Value of Criterion

Selection stopped because the change of the maximum absolute correction is tiny.





The selected model, based on Validation ASE, is the model at Step 13.

Effects: Intercept Lot_Area Gr_Liv_Area Garage_Area Basement_Area Deck_Porch_Area Age_Sold
Heating_QC_Ex Heating_QC_Fa Heating_QC_TA Central_Air_N Fireplaces_0 Fireplaces_3
Lot_Shape_2_Irregular

Analysis of Variance				
Source	DF	Sum of Squares	Mean Square	F Value
Model	13	7.244182E11	55724479151	183.41
Error	660	2.005289E11	303831708	
Corrected Total	673	9.249472E11		
Root MSE		17431		
Dependent Mean		140648		
R-Square		0.7832		
Adj R-Sq		0.7789		
AIC		13854		
AICC		13855		
SBC		13242		
ASE (Train)		297520664		
ASE (Validate)		290198671		

Parameter Estimates

Parameter	DF	Estimate
Intercept	1	62180
Lot_Area	1	0.672772
Gr_Liv_Area	1	49.273408
Garage_Area	1	28.268178
Basement_Area	1	29.532418
Deck_Porch_Area	1	25.246979
Age_Sold	1	-439.999571
Heating_QC_Ex	1	3362.383978
Heating_QC_Fa	1	-11789
Heating_QC_TA	1	-8849.795097
Central_Air_N	1	-2506.032791
Fireplaces_0	1	-7780.500792
Fireplaces_3	1	36383
Lot_Shape_2_Irregular	1	5348.064492

2. The data set **Newdata_Ames_Reg** contains five new observations from the Ames housing market.
Use PROC PLM to score the new data set.

```
proc plm restore=mymod;
  score data=sp4r.newdata_ames_reg out=sp4r.pred_newdata
    predicted;
run;
```

3. Print the predicted values and the intervals from the new scored data set along with the response and the independent variables.

```
proc print data=sp4r.pred_newdata;
  var saleprice predicted;
run;
```

Obs	SalePrice	Predicted
1	213500	179006.18
2	191500	179438.13
3	115000	111322.54
4	160000	166195.97
5	180000	169357.57

End of Demonstration



Polynomial Regression with PROC GLMSELECT

SP4R06d06.sas

- Recall that the previous polynomial regression model used the **Paper** data set to create a model with **Strength** as the dependent variable and **Amount** of chemical additive as the independent variable. You used a linear, quadratic, and cubic term to model the strength of the paper. This time, use the EFFECT statement to create a polynomial model to degree 5. Use the forward selection method with an entry barrier of 0.05 for a significance level. In addition, use the HIERARCHY=SINGLE option in the MODEL statement to enforce polynomial hierarchy. Use the OUTDESIGN= option to create a new data table with the new polynomial variables.

```
proc glmselect data=sp4r.paper outdesign=sp4r.des;
  effect amount_pol = polynomial(amount / degree=5);
  model strength = amount_pol / selection=forward select=s1 sle=.05
    hierarchy=single;
run;quit;
```

The GLMSELECT Procedure

Data Set	WORK.PAPER
Dependent Variable	Strength
Selection Method	Forward
Select Criterion	Significance Level
Stop Criterion	Significance Level
Entry Significance Level (SLE)	0.05
Effect Hierarchy Enforced	Single
Number of Observations Read	22
Number of Observations Used	22

Dimensions

Number of Effects	6
Number of Parameters	6

Forward Selection Summary

Step	Effect Entered	Number Effects In	F Value	Pr > F
0	Intercept	1	0.00	1.0000
1	Amount	2	24.08	<.0001
2	Amount^2	3	5.90	0.0253
3	Amount^3	4	5.08	0.0369

Selection stopped as the candidate for entry has SLE > 0.05.

Stop Details

Candidate For	Effect	Candidate Significance	Compare Significance
Entry	Amount^4	0.3682	> 0.0500 (SLE)

The selected model is the model at the last step (Step 3).

Effects: Intercept Amount Amount^2 Amount^3

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value
Model	3	0.52986	0.17662	16.22
Error	18	0.19605	0.01089	
Corrected Total	21	0.72591		
Root MSE		0.10436		
Dependent Mean		2.81364		
R-Square		0.7299		
Adj R-Sq		0.6849		
AIC		-71.84939		
AICC		-68.09939		
SBC		-91.48522		

Parameter Estimates

Parameter	DF	Estimate	Standard Error	t Value
Intercept	1	2.732803	0.260598	10.49
Amount	1	-0.368996	0.322081	-1.15
Amount^2	1	0.223389	0.116506	1.92
Amount^3	1	-0.028619	0.012699	-2.25

2. Print the new data table to ensure that the quadratic and cubic terms were created correctly. Then, use the &_GLSMOD macro variable.

```
proc print data=sp4r.des;
run;
```

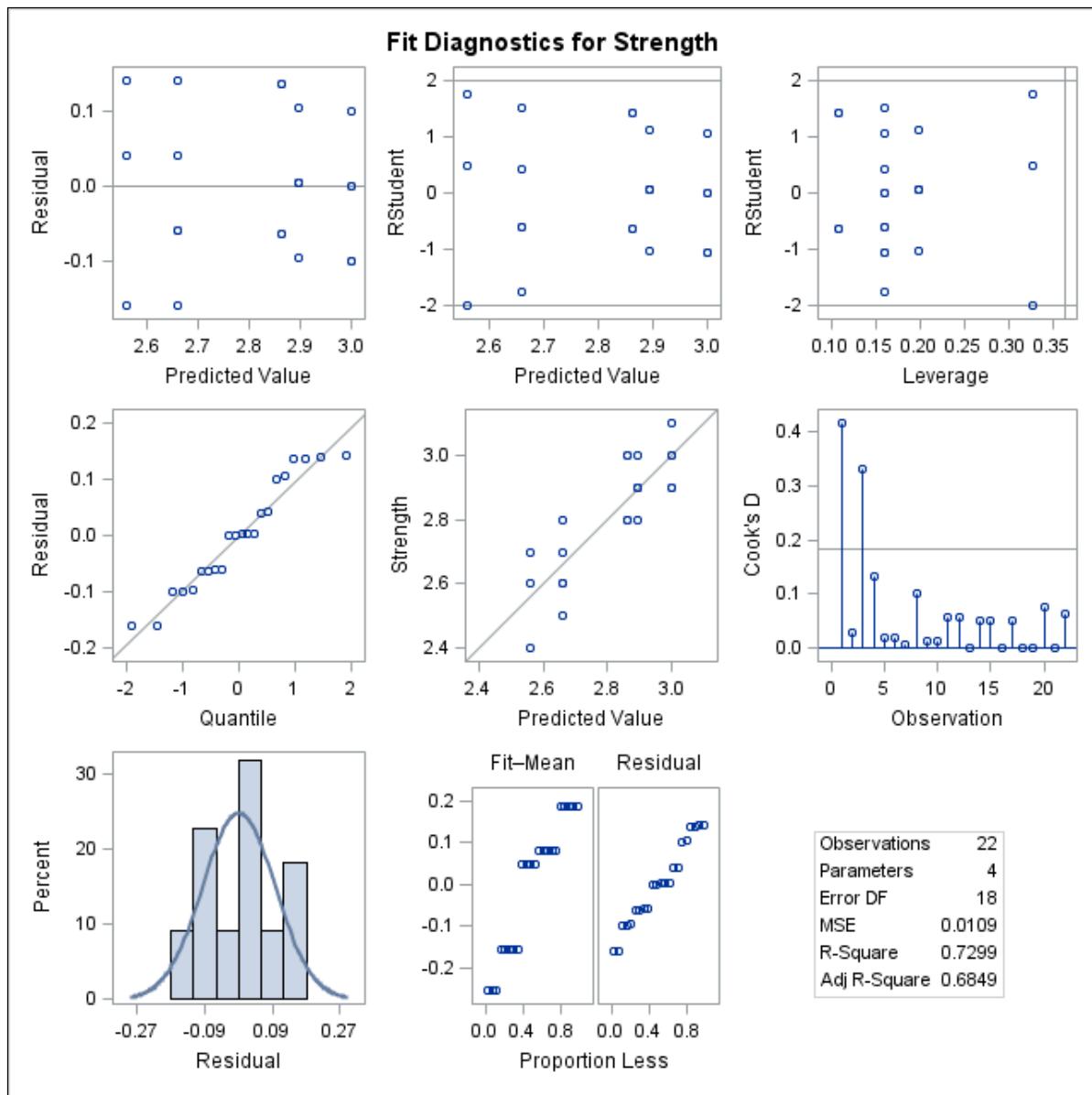
Obs	Intercept	Amount	Amount_2	Amount_3	Strength
1	1	1	1	1	2.4
2	1	1	1	1	2.6
3	1	1	1	1	2.7
4	1	2	4	8	2.5
5	1	2	4	8	2.6
6	1	2	4	8	2.6
7	1	2	4	8	2.7
8	1	2	4	8	2.8
9	1	3	9	27	2.8
10	1	3	9	27	2.8
11	1	3	9	27	3.0
12	1	3	9	27	3.0
13	1	4	16	64	3.0
14	1	4	16	64	2.9
15	1	4	16	64	2.9
16	1	4	16	64	3.0
17	1	4	16	64	3.1
18	1	5	25	125	2.9

19	1	5	25	125	2.9
20	1	5	25	125	3.0
21	1	5	25	125	2.9
22	1	5	25	125	2.8

```
proc reg data=sp4r.des;
  model strength = &_glsmod;
run;quit;
```

Partial Output

The REG Procedure					
Model: MODEL1					
Dependent Variable: Strength					
Number of Observations Read					22
Number of Observations Used					22
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	0.52986	0.17662	16.22	<.0001
Error	18	0.19605	0.01089		
Corrected Total	21	0.72591			
Root MSE		0.10436	R-Square	0.7299	
Dependent Mean		2.81364	Adj R-Sq	0.6849	
Coeff Var		3.70919			
Parameter Estimates					
Variable	Label	DF	Parameter Estimate	Standard Error	t Value
Intercept	Intercept	1	2.73280	0.26060	10.49
Amount	Amount	1	-0.36900	0.32208	-1.15
Amount_2	Amount^2	1	0.22339	0.11651	1.92
Amount_3	Amount^3	1	-0.02862	0.01270	-2.25
					0.0369



End of Demonstration

6.2 Generalized Linear Models

Objectives

- Use PROC LOGISTIC to create logistic regression models.
- Use PROC GENMOD to create generalized linear models with a link other than the logit link.

52

Motivation

Use a PROC step to perform the following tasks:

- create generalized linear models
- create statistical graphics
- save important model information

$$g(E(Y)) = g(\mu) = X\beta$$



53

Duplicating the R Script

```

#Logistic
mylm = glm(Bonus ~ Basement_Area, family=binomial)
anova(mylm); summary(mylm)
par(mfrow=c(2,2)); plot(mylm)

#Poisson
mylm = glm(satellites ~ weight, family=poisson)
anova(mylm); summary(mylm)
par(mfrow=c(2,2)); plot(mylm)

#Prediction
predict(mylm, newdata=mynew.dataframe)

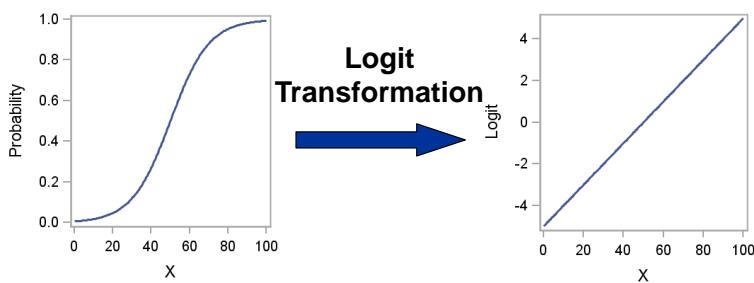
```

54

Logistic Regression

Recall that logistic regression uses the logit transformation to model binary variables.

$$\text{logit}(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 X_{1i} + \dots + \beta_k X_{ki}$$



55

The assumption in logistic regression is that the logit has a linear relationship with the predictor variables.

LOGISTIC Procedure

Use PROC LOGISTIC to conduct a logistic regression.

```

Source on Save | Run | Source
mylm = glm(Bonus ~ Basement_Area, family=binomial)
anova(mylm); summary(mylm)
par(mfrow=c(2,2)); plot(mylm)

proc logistic data=ameshousuing;
  model bonus(event='1') = basement_area;
run;

PROC LOGISTIC DATA=data-table-name <options>;
  MODEL dependent-variable(EVENT=) = effects;
RUN;

```

56

Selected PROC LOGISTIC MODEL statement and options:

- MODEL specifies the response variable followed by an equal symbol and the explanatory effects.
 EVENT specifies the event category for the binary response model.

LOGISTIC Procedure

View the default output.

```

Source on Save | Run | Source
mylm = glm(Bonus ~ Basement_Area, family=binomial)
anova(mylm); summary(mylm)
par(mfrow=c(2,2)); plot(mylm)

proc logistic data=ameshousuing;
  model bonus(event='1') = basement_area;
run;

```

Model Fit Statistics		
Criterion	Intercept Only	Intercept and Covariates
AIC	255.625	161.838
SC	259.329	169.246
-2 Log L	253.625	157.838

Testing Global Null Hypothesis: BETA=0				
Test	Chi-Square	DF	Pr > ChiSq	
Likelihood Ratio	95.7870	1	<.0001	
Score	65.5624	1	<.0001	
Wald	48.0617	1	<.0001	

57

The value of 157.838 is the same as the residual deviance in R and the value of 95.78 is the same as the deviance.

LOGISTIC Procedure

View the default output.

```
Source on Save | Run | Source
my1m = glm(Bonus ~ Basement_Area, family=binomial)
anova(my1m); summary(my1m)
par(mfrow=c(2,2)); plot(my1m)

proc logistic data=ameshousuing;
  model bonus(event='1') = basement_area;
run;
```

Odds Ratio Estimates			
Effect	Point Estimate	95% Wald Confidence Limits	
		Lower	Upper
Basement_Area	1.007	1.005	1.010

Analysis of Maximum Likelihood Estimates					
Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-9.7854	1.2896	57.5758	<.0001
Basement_Area	1	0.00739	0.00107	48.0617	<.0001

58

LOGISTIC Procedure

Use the UNITS statement to specify the units of change for a continuous variable and the CLODDS= option in the MODEL statement to request a confidence interval.

```
proc logistic data=ameshousuing;
  model bonus(event='1') = basement_area
    /cloodds=wald;
  units basement_area=100;
run;
```

Odds Ratio Estimates and Wald Confidence Intervals				
Effect	Unit	Estimate	95% Confidence Limits	
Basement_Area	100.0	2.095	1.700	2.582

59

Selected PROC LOGISTIC statements and options:

- UNITS specifies the units of change for the continuous explanatory variable so that customized odds ratios can be estimated. The UNITS statement is ignored for CLASS variables. Odds ratios are computed for only those main effects that are not involved in interactions or nestings, unless the ODDSRATIO statement is also specified.
- CLOODDS produces confidence intervals for odds ratios of main effects not involved in interactions or nestings. Computation of these confidence intervals is based on the profile likelihood (CLOODDS=PL) or individual Wald tests (CLOODDS=WALD).

LOGISTIC Procedure

Other default output:

Association of Predicted Probabilities and Observed Responses			
Percent Concordant	89.5	Somers' D	0.791
Percent Discordant	10.4	Gamma	0.792
Percent Tied	0.1	Tau-a	0.202
Pairs	11475	c	0.896

- Counting concordant, discordant, and tied pairs is another tool that is used for model assessment.

60

The Association of Predicted Probabilities and Observed Responses table lists several measures of association to help you assess the predictive ability of the logistic model.

The number of pairs used to calculate the values of this table is equal to the product of the counts of observations with positive responses and negative responses. In this example, the value is $255 \times 45 = 11,475$.

Comparing Pairs

To find concordant, discordant, and tied pairs, compare houses that had the outcome of interest against houses that did not.



$$255 \text{ Bonus Eligible} * 45 \text{ Not Bonus Eligible} = 11,475 \text{ pairs}$$

61

Concordant Pair

Compare a 1200-square-foot basement that was bonus eligible with an 800-square-foot basement that was not.

Not Eligible, 800 sq ft



$P(\text{Eligible})=.0204$

Bonus Eligible, 1200 sq ft



$P(\text{Eligible})=.2865$

The actual sorting agrees with the model.

This is a **concordant** pair.

62

For all pairs of observations with different values of the response variable, a pair is **concordant** if the observation with the outcome has a higher predicted outcome probability (based on the model) than the observation without the outcome.

Discordant Pair

Compare a 1400-square-foot basement that was bonus eligible with a 1600-square-foot basement that was not.

Not Eligible, 1600 sq ft



$P(\text{Eligible})=.8855$

Bonus Eligible, 1400 sq ft



$P(\text{Eligible})=.6379$

The actual sorting disagrees with the model.

This is a **discordant** pair.

63

A pair is **discordant** if the observation with the outcome has a lower predicted outcome probability than the observation without the outcome.

6.04 Multiple Choice Poll

Consider a logistic regression model where the binary response is whether a person is a dog owner. You sample 140 people and find that 95 people are dog owners. How many total concordant and discordant pairs are considered?

- a. 140
- b. $95 \times 45 = 4275$
- c. $(95 \times 45) / 140 = 30.53$
- d. $95 \times 45 \times 140 = 598500$

64

LOGISTIC Procedure

Stepwise selection for logistic regression models is specified in the PROC LOGISTIC MODEL statement.

- Use the SELECTION= option to specify FORWARD, BACKWARD, or STEPWISE model selection.

Default selection criteria:

	PROC REG/ PROC GLMSELECT		PROC LOGISTIC	
	SLENTRY	SLSTAY	SLENTRY	SLSTAY
FORWARD	0.50	-----	0.05	
BACKWARD	-----	0.10		0.05
STEPWISE	0.15	0.15	0.05	0.05

66



Logistic Regression

SP4R06d07.sas

Use the **AmesHousing** data set to create a logistic regression model.

1. Select **Bonus** as the outcome variable and use **Basement_Area**, **Fireplaces**, and **Lot_Shape_2** as the predictor variables. Specify reference cell coding for the categorical variables and specify *0* as the reference group for **Fireplaces** and *Regular* for **Lot_Shape_2**. Use the EVENT=1 option to model the probability of a house being greater than \$175,000. Use the UNITS statement to identify the odds ratio for an increase in **Basement_Area** of 100 square feet. Make sure that you specify the CLODDS=WALD option to request a confidence interval. Use an ESTIMATE statement to compute the log odds of **Bonus** for a house with a basement area of 1000 square feet, one fireplace, and irregular **Lot_Shape_2**. Use an OUTPUT statement to save the predicted probabilities. Use a STORE statement to save the model.

```
proc logistic data=sp4r.ameshousing
  plots(only)=(effect oddsratio roc);
  class fireplaces(ref='0') lot_shape_2(ref='Regular') / param=ref;
  model bonus(event='1') = basement_area fireplaces lot_shape_2
    / clodds=wald;
  units basement_area = 100;
  estimate 'my estimate' intercept 1 basement_area 1000
    fireplaces 1 0 lot_shape_2 1 / e alpha=.05 ilink;
  output out=sp4r.out p=pred;
  store mymod;
run;
```

Selected PROC LOGISTIC options:

- | | |
|-------------|---|
| PLOTS(ONLY) | specifies which plots to display. |
| EFFECT | displays and enhances the effect plots of the model. |
| ODDSRATIO | displays and enhances the odds ratio plots for the model when the CLODDS=option or the ODDSRATIO statement is also specified. |
| ROC | displays the ROC curve. |
| ILINK | computes the estimate on the original data scale. |
| P | requests the predicted probabilities (based on the model) to be included in the output data table. |

The LOGISTIC Procedure

Model Information

Data Set	WORK.AMESHOUSING
Response Variable	Bonus
Number of Response Levels	2
Model	binary logit
Optimization Technique	Fisher's scoring
Number of Observations Read	300
Number of Observations Used	299
Response Profile	

Ordered Value	Bonus	Total Frequency
1	0	255
2	1	44

Probability modeled is Bonus='1'.

NOTE: 1 observation was deleted due to missing values for the response or explanatory variables.

Class Level Information

Class	Value	Design Variables	
Fireplaces	0	0	0
	1	1	0
	2	0	1
Lot_Shape_2	Irregular	1	
	Regular	0	

Model Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Intercept Only	Intercept and Covariates	Intercept
AIC	251.812	140.499	
SC	255.513	159.001	
-2 Log L	249.812	130.499	

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	119.3133	4	<.0001
Score	91.7250	4	<.0001
Wald	49.8671	4	<.0001

Type 3 Analysis of Effects

Effect	DF	Chi-Square	Wald	Pr > ChiSq
Basement_Area	1	38.1356		<.0001
Fireplaces	2	5.2060		0.0741
Lot_Shape_2	1	16.9421		<.0001

Analysis of Maximum Likelihood Estimates

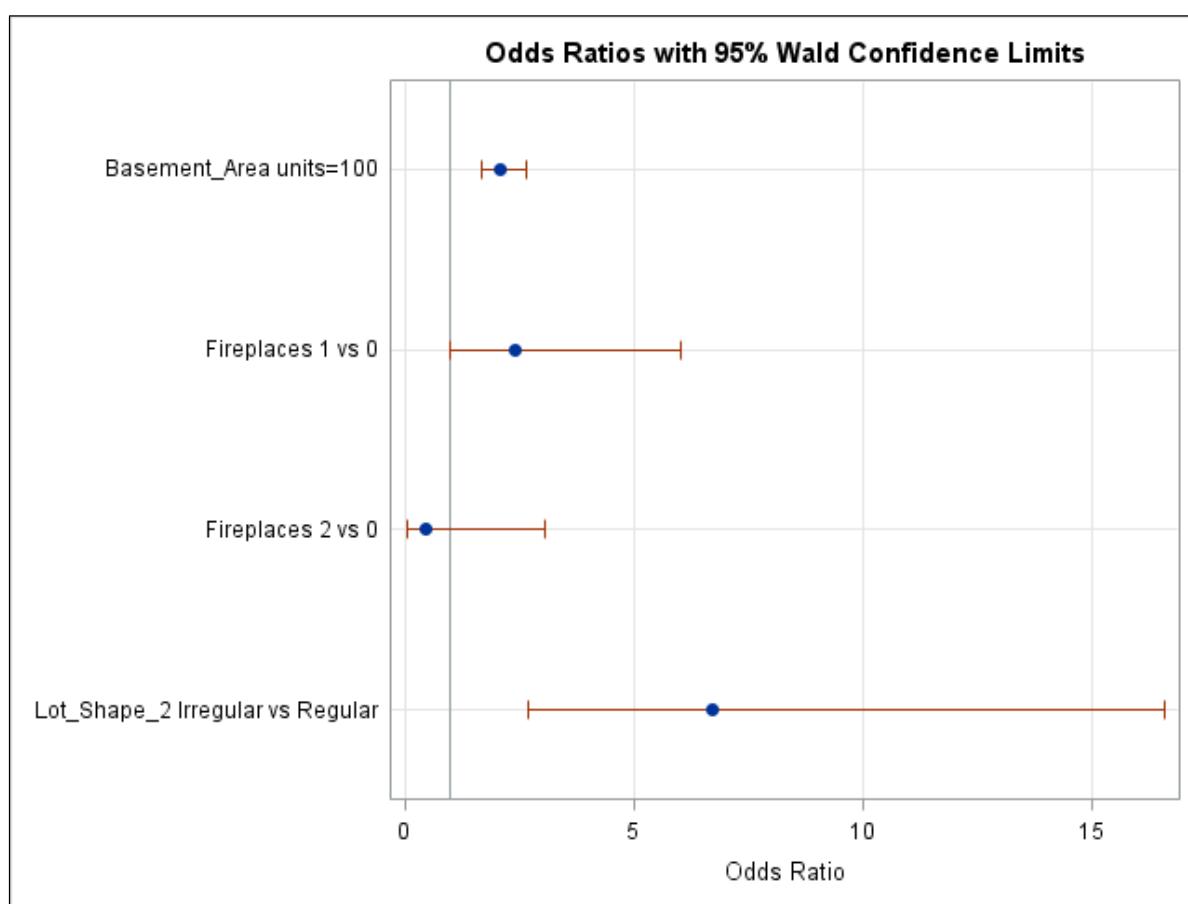
Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-11.0882	1.5384	51.9467	<.0001
Basement_Area	1	0.00744	0.00120	38.1356	<.0001
Fireplaces 1	1	0.8810	0.4658	3.5770	0.0586
Fireplaces 2	1	-0.7683	0.9654	0.6335	0.4261
Lot_Shape_2 Irregular	1	1.9025	0.4622	16.9421	<.0001

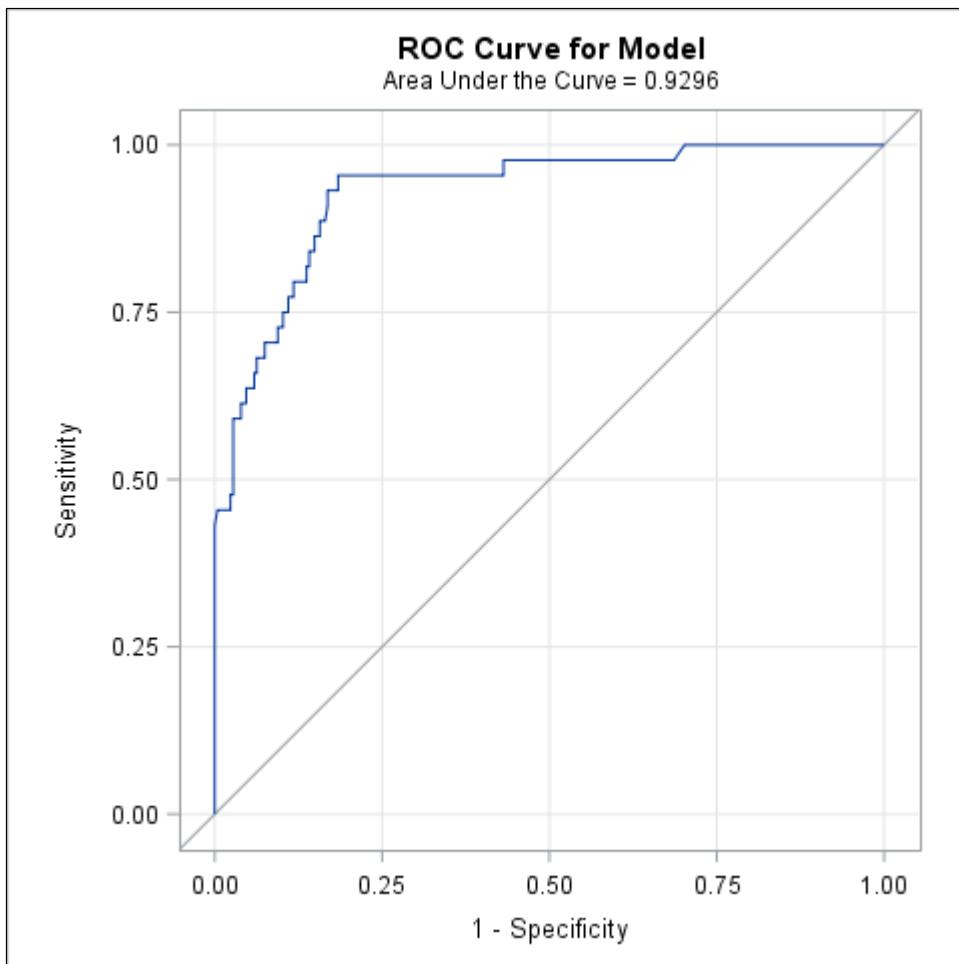
Association of Predicted Probabilities and Observed Responses

Percent Concordant	92.9	Somers' D	0.859
Percent Discordant	7.0	Gamma	0.860
Percent Tied	0.1	Tau-a	0.216
Pairs	11220	c	0.930

Odds Ratio Estimates and Wald Confidence Intervals

Effect	Unit	Estimate	95% Confidence Limits
Basement_Area	100.0	2.105	1.662 2.665
Fireplaces 1 vs 0	1.0000	2.413	0.969 6.014
Fireplaces 2 vs 0	1.0000	0.464	0.070 3.076
Lot_Shape_2 Irregular vs Regular	1.0000	6.703	2.709 16.584



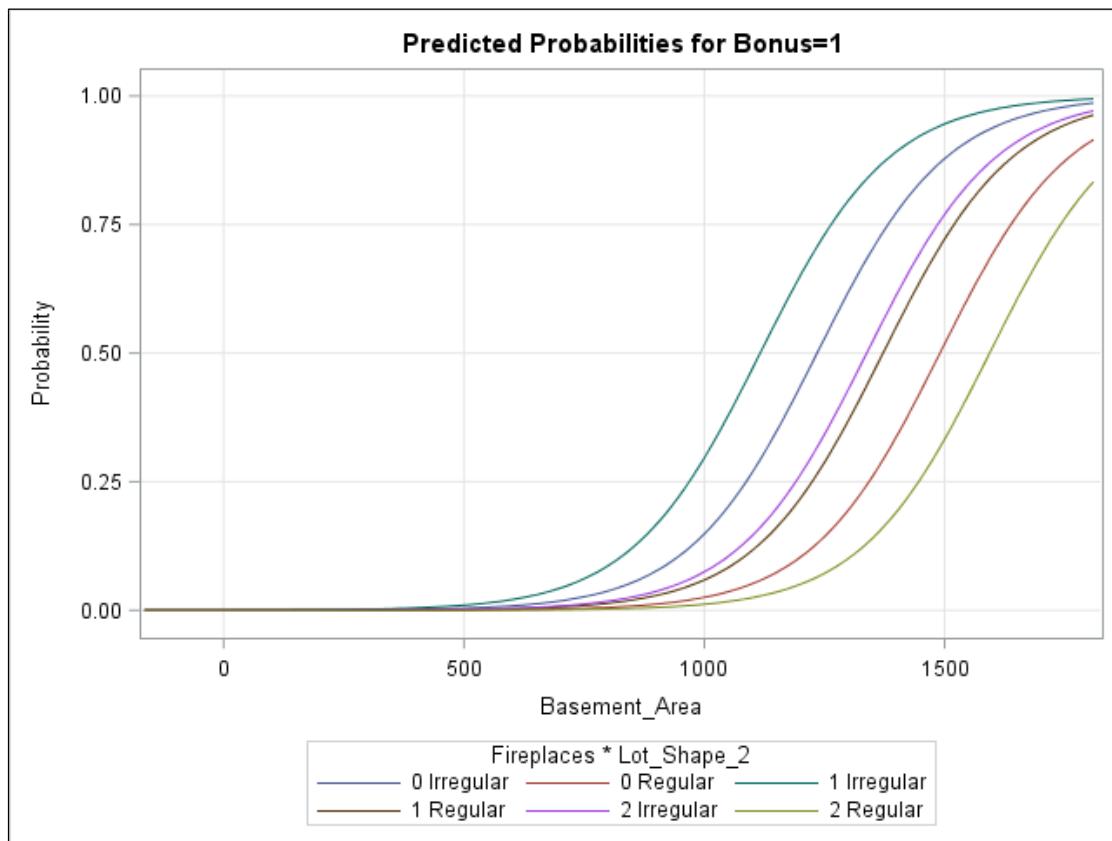


Recall that the Sensitivity measures the proportion of positives correctly identified and the Specificity measures the proportion of negatives correctly identified. Ideally, the area under the curve is near 1.

Estimate Coefficients									
			Lot_Shape_						
Parameter	2		Fireplaces		Row1				
Intercept: Bonus=0									1
Basement_Area									1000
Fireplaces 1						1			1
Fireplaces 2						2			
Lot_Shape_2 Irregular			Irregular						1

Estimate									
Label	Estimate	Standard					Standard Error of		
		Error	z	Value	Pr > z	Alpha	Lower	Upper	Mean
my estimate	-0.8636	0.4068	-2.12	0.0338	0.05	-1.6609	-0.06633	0.2966	0.08486
Estimate									

Label	Lower Mean	Upper Mean
my estimate	0.1596	0.4834



```
proc print data=sp4r.out (obs=5);
  var bonus basement_area fireplaces lot_shape_2 pred;
run;
```

Obs	Bonus	Basement_Area	Fireplaces	Lot_Shape_2	pred
1	1	1338	0	Regular	0.24377
2	1	1280	0	Irregular	0.58391
3	0	864	1	Regular	0.02235
4	0	1145	0	Regular	0.07121
5	1	384	1	Irregular	0.00429

2. The data set **Newdata_Ames_Logistic** contains five new homes to score in the Ames housing market. Use PROC PLM to score the data and request the predicted values and confidence limits. Be sure to use the ILINK option to predict on the probability scale.

```
proc plm restore=mymod;
  score data=sp4r.newdata_ames_logistic out=sp4r.pred_newdata
    predicted lclm uclm /ilink;
run;
```

The PLM Procedure

Store Information

Item Store	WORK.MYMOD
Data Set Created From	WORK.AMESHOUSING
Created By	PROC LOGISTIC
Date Created	29NOV15:12:47:14
Response Variable	Bonus
Link Function	Logit
Distribution	Binary
Class Variables	Fireplaces Lot_Shape_2 Bonus
Model Effects	Intercept Basement_Area Fireplaces Lot_Shape_2

3. Print the scored data and interval information as well as the response and independent variables.

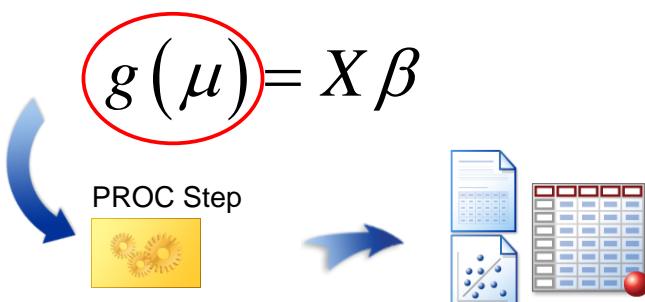
```
proc print data=sp4r.pred_newdata;
  var bonus basement_area fireplaces lot_shape_2 predicted lclm uclm;
run;
```

Obs	Bonus	Area	Basement_Fireplaces		Predicted	Lot_Shape_	
			2	1		LCLM	UCLM
1	1	1019		1	0.32690	0.18195	0.51468
2	0	845		0	0.00816	0.00245	0.02686
3	0	1008		0	0.15642	0.07340	0.30268
4	0	576		0	0.00111	0.00019	0.00629
5	0	637		0	0.00175	0.00035	0.00871

End of Demonstration

Motivation

Use PROC GENMOD to specify the distribution or use the LINK= function to create a generalized linear model.



68

Female Horseshoe Crab Example



Number of Satellites

- Color
- Spine condition
- Carapace width in centimeters
- Weight in grams

69

The data come from a study that was conducted about the mating habits of female horseshoe crabs. The population of horseshoe crabs is monitored because they provide a critical food source for migrating birds. Each year, at the end of May and during June, hundreds of thousands of horseshoe crabs emerge from Delaware Bay to lay and fertilize their eggs.

Each female horseshoe crab had a male crab resident in her nest. The study investigated factors affecting whether the female horseshoe crab had any other males, called *satellites*, residing nearby. The response variable for each female horseshoe crab is her number of satellites. The data are stored in **Crab**.

The data are from an example in Agresti (1996).

The Data

Width	Weight	Color	Spine	Satellites
28.3	3.05	2	3	8
22.5	1.55	3	3	0
26.0	2.30	1	1	9
24.8	2.10	3	3	0
26.0	2.60	3	3	4
23.8	2.10	2	3	0
26.5	2.35	1	1	0
24.7	1.90	3	2	0
23.7	1.95	2	1	0
25.6	2.15	3	3	0
24.3	2.15	3	3	0
25.8	2.65	2	3	0
28.2	3.05	2	3	11
21.0	1.85	4	2	0
26.0	2.30	2	1	14
27.1	2.95	1	1	8
...				

Color: 1=Light Medium 2=Medium 3=Dark Medium 4=Dark

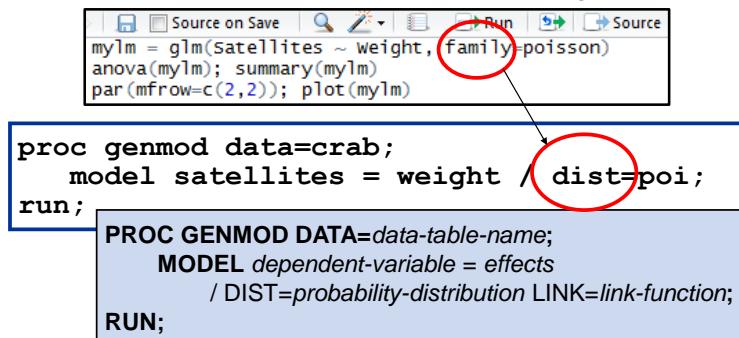
Spine: 1=Both Good 2=One Worn or Broken 3=Both Worn or Broken

70

The response variable (**Satellites**) is a count variable.

GENMOD Procedure

Use PROC GENMOD to conduct Poisson regression.



```

Source on Save Run Source
mylm = glm(satellites ~ weight, family=poisson)
anova(mylm); summary(mylm)
par(mfrow=c(2,2)); plot(mylm)

proc genmod data=crab;
  model satellites = weight / dist=poi;
run;

PROC GENMOD DATA=data-table-name;
  MODEL dependent-variable = effects
    / DIST=probability-distribution LINK=link-function;
RUN;

```

The code block shows two parts: an R session at the top and SAS code at the bottom. The R session contains a glm call with family=poisson and some post-fit analysis. The SAS code shows a proc genmod statement with a model statement containing a / dist=poi option, and a PROC GENMOD block with its own syntax for specifying distribution and link functions.

Default output:

- parameter estimates
- goodness of fit assessment

71

Selected PROC GENMODEL MODEL statement options:

DIST specifies the distribution of the response variable.

LINK specifies the desired link function to be used for analysis.

GENMOD Procedure

Specify the DIST= option in the MODEL statement.

DIST=	Distribution	Default Link Function
BINOMIAL	Binomial	Logit
GAMMA	Gamma	Inverse
GEOMETRIC	Geometric	Log
IGAUSSIAN	Inverse Gaussian	Inverse squared
MULTINOMIAL	Multinomial	Cumulative logit
NEGBIN	Negative Binomial	Log
NORMAL	Normal	Identity
POISSON	Poisson	Log
ZIP	Zero-inflated Poisson	Log/Logit
ZINB	Zero-inflated Negative Binomial	Log/Logit

72

The default link function is used if no LINK= option is specified by the user.

The DIST=BINOMIAL option provides the same analysis as PROC LOGISTIC.

GENMOD Procedure

As an alternative, specify the LINK= option in the MODEL statement.

LINK=	Link Function
CUMLL, CCLL	Cumulative Complementary Log-Log
CUMLOGIT, CLOGIT	Cumulative Logit
CUMPROBIT, CPROBIT	Cumulative Probit
CLOGLOG, CLL	Complementary Log-Log
IDENTITY, ID	Identity
LOG	Log
LOGIT	Logit
PROBIT	Probit
POWER	Power

73

The cumulative LINK functions are appropriate only for the multinomial distribution.

6.05 Poll

Logistic regression can be conducted in both PROC LOGISTIC and PROC GENMOD.

- True
- False



GENMOD Procedure

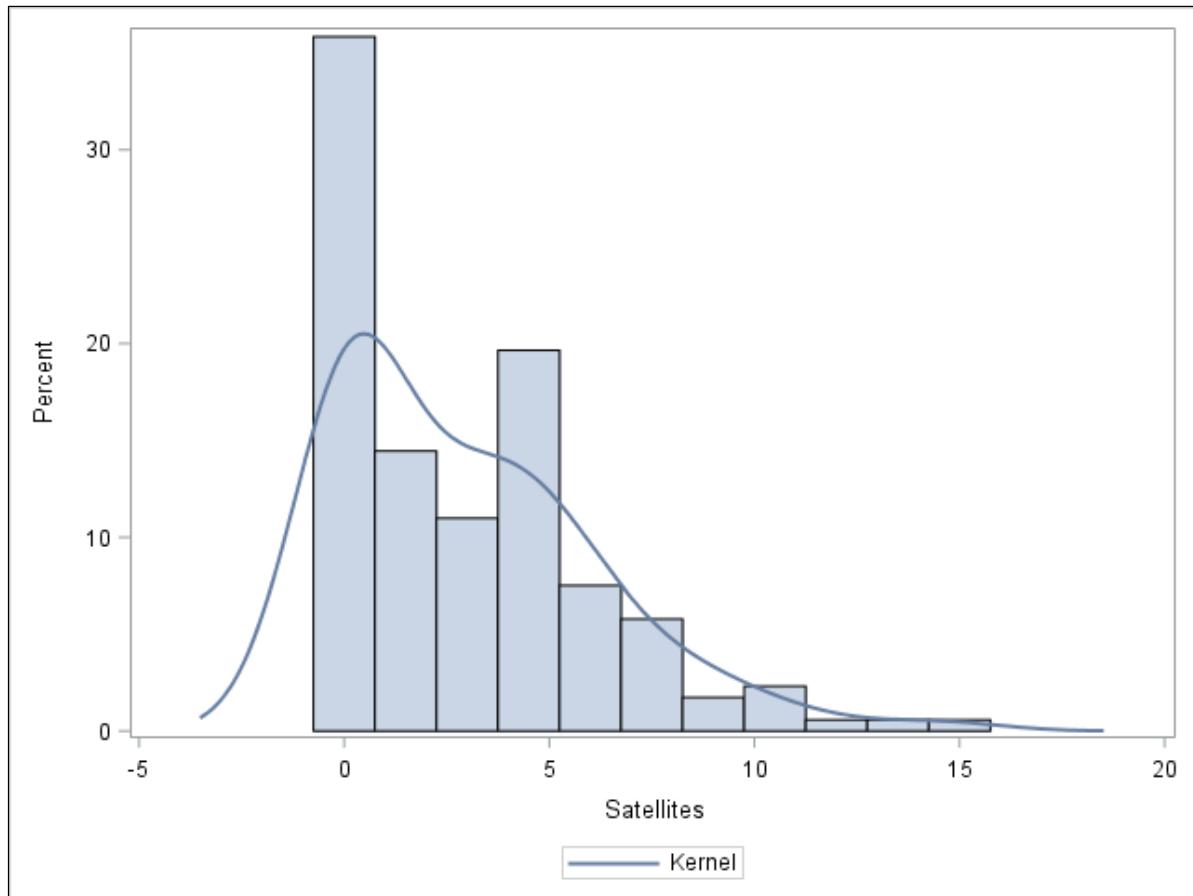
SP4R06d08.sas

1. Create a histogram with a kernel density estimate for the **Satellites** variable from the **Crab** data set. Use PROC MEANS to analyze the continuous variables **Satellites**, **Width**, and **Weight**. Use PROC FREQ to analyze the variables, **Satellites**, **Color**, and **Spine**.

```
proc sgplot data=sp4r.crab;
  histogram satellites;
  density satellites / type=kernel;
run;

proc means data=sp4r.crab;
  var satellites width weight;
run;

proc freq data=sp4r.crab;
  tables satellites color spine;
run;
```



The MEANS Procedure					
Variable	N	Mean	Std Dev	Minimum	Maximum
Satellites	173	2.9190751	3.1483357	0	15.0000000
Width	173	26.2988439	2.1090610	21.0000000	33.5000000
Weight	173	2.4371908	0.5770252	1.2000000	5.2000000

The FREQ Procedure					
Satellites	Frequency	Percent	Cumulative Frequency	Cumulative Percent	
0	62	35.84	62	35.84	
1	16	9.25	78	45.09	
2	9	5.20	87	50.29	
3	19	10.98	106	61.27	
4	19	10.98	125	72.25	
5	15	8.67	140	80.92	
6	13	7.51	153	88.44	
7	4	2.31	157	90.75	
8	6	3.47	163	94.22	
9	3	1.73	166	95.95	
10	3	1.73	169	97.69	
11	1	0.58	170	98.27	
12	1	0.58	171	98.84	
14	1	0.58	172	99.42	
15	1	0.58	173	100.00	

Color	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	12	6.94	12	6.94
2	95	54.91	107	61.85
3	44	25.43	151	87.28
4	22	12.72	173	100.00

Spine	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	37	21.39	37	21.39
2	15	8.67	52	30.06
3	121	69.94	173	100.00

2. Use the **Crab** data set to create a Poisson regression model with PROC GENMOD. Be sure to specify **DIST=POI** as the MODEL statement option. Let the number of **Satellites** be the dependent variables and let the independent variables be **Width**, **Weight**, **Color**, and **Spine**. Use the CLASS statement to specify reference cell coding and specify **1** as the reference group for both **Color** and **Spine**. Use the ESTIMATE statement to identify the expected number of **Satellites** for a crab with a width of 25, and a weight of 2.5, a color category of 2, and a spine category of 2. Finally, use the OUTPUT statement to create a data table in order to create a residual versus predicted plot.

```

proc genmod data=sp4r.crab plots=resraw;
  class color(param=ref ref='1') spine(param=ref ref='1');
  model satellites = width weight color spine
    / dist=poi link=log type3;
  estimate 'my estimate' intercept 1 width 25 weight 2.5
    color 1 0 0 spine 1 0 / e exp alpha=.05;
  output out=sp4r.out p=pred resraw=res;
run;

```

The GENMOD Procedure

Model Information

Data Set	WORK.CRAB
Distribution	Poisson
Link Function	Log
Dependent Variable	Satellites

Number of Observations Read	173
Number of Observations Used	173

Class Level Information

Class	Value	Design Variables		
Color	1	0	0	0
	2	1	0	0
	3	0	1	0
	4	0	0	1
Spine	1	0	0	
	2	1	0	
	3	0	1	

Parameter Information

Parameter	Effect	Color	Spine
Prm1	Intercept		
Prm2	Width		
Prm3	Weight		
Prm4	Color	2	
Prm5	Color	3	
Prm6	Color	4	
Prm7	Spine		2
Prm8	Spine		3

Criteria For Assessing Goodness Of Fit

Criterion	DF	Value	Value/DF
Deviance	165	549.5856	3.3308
Scaled Deviance	165	549.5856	3.3308
Pearson Chi-Square	165	533.8165	3.2353
Scaled Pearson X2	165	533.8165	3.2353
Log Likelihood		77.5928	
Full Log Likelihood		-452.4416	

Criteria For Assessing Goodness Of Fit							
Criterion		DF	Value		Value/DF		
AIC (smaller is better)			920.8833				
AICC (smaller is better)			921.7613				
BIC (smaller is better)			946.1096				

Algorithm converged.

Analysis Of Maximum Likelihood Parameter Estimates							
Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits	Chi-Square	Wald	Pr > ChiSq
Intercept	1	-0.3618	0.9666	-2.2562 1.5326	0.14	0.7082	
Width	1	0.0167	0.0489	-0.0791 0.1126	0.12	0.7321	
Weight	1	0.4965	0.1663	0.1706 0.8223	8.92	0.0028	
Color	2	1 -0.2649	0.1681	-0.5943 0.0646	2.48	0.1152	
Color	3	1 -0.5137	0.1954	-0.8966 -0.1308	6.91	0.0086	
Color	4	1 -0.5309	0.2269	-0.9756 -0.0861	5.47	0.0193	
Spine	2	1 -0.1504	0.2136	-0.5690 0.2682	0.50	0.4814	
Spine	3	1 0.0873	0.1199	-0.1478 0.3223	0.53	0.4667	
Scale	0	1.0000	0.0000	1.0000 1.0000			

NOTE: The scale parameter was held fixed.

LR Statistics For Type 3 Analysis							
Source	DF	Chi-Square	Pr > ChiSq				
Width				1	0.12	0.7324	
Weight				1	9.04	0.0026	
Color				3	9.24	0.0263	
Spine				2	1.79	0.4076	

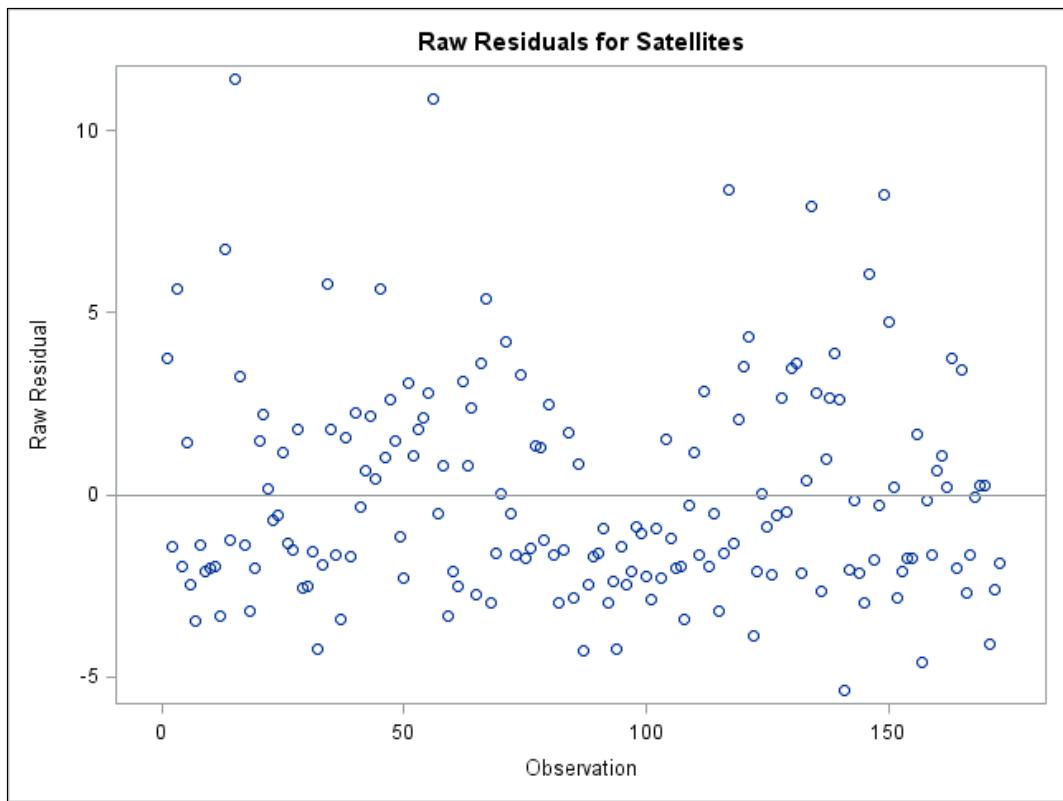
Coefficients for Contrast my estimate								
Label	Prm1	Prm2	Prm3	Prm4	Prm5	Prm6	Prm7	Prm8
my estimate	1	25	2.5	1	0	0	1	0

Contrast Estimate Results

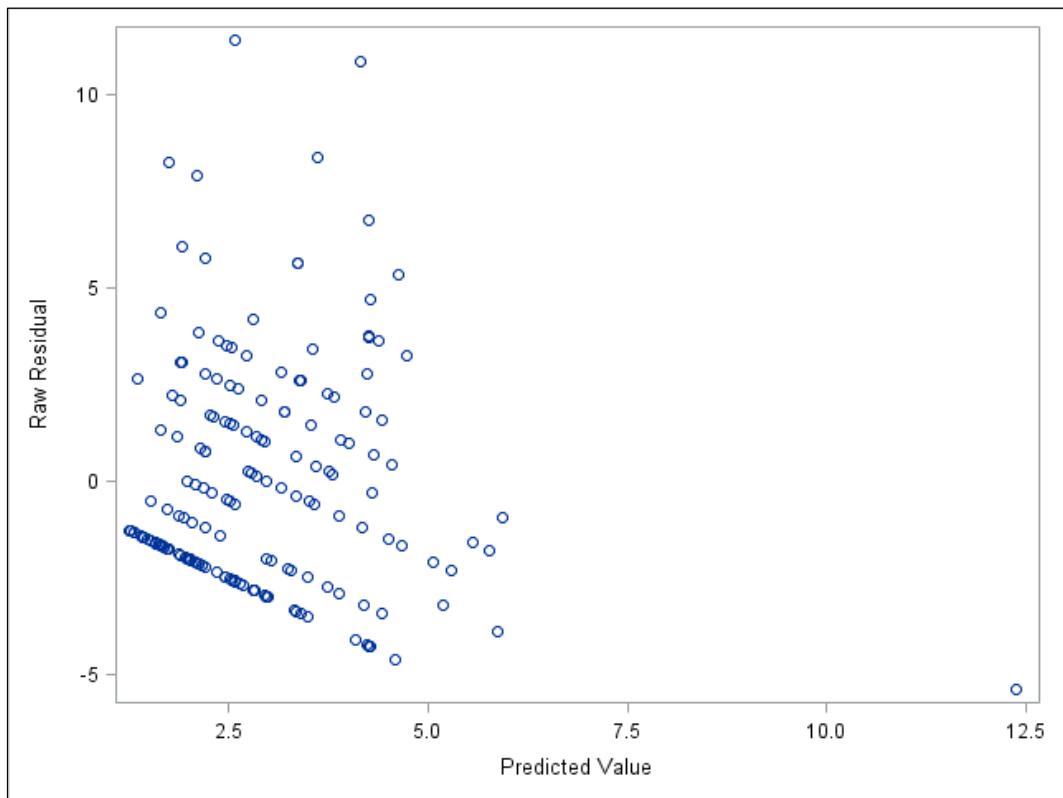
Label	Mean Estimate	Mean Confidence Limits	L'Beta Estimate	Standard Error	Alpha	L'Beta Confidence Limits
my estimate	2.4178	1.6538 3.5349	0.8829	0.1938	0.05	0.5030 1.2627
Exp(my estimate)			2.4178	0.4686	0.05	1.6538 3.5349

Contrast Estimate Results

Label	Chi-Square	Pr > ChiSq
my estimate	20.76	<.0001
Exp(my estimate)		



```
proc sgplot data=sp4r.out;
  scatter y=res x=pred;
run;
```



3. Poisson regression models assume that the variance is equal to the mean. However, count data often exhibit variability that exceeds the mean. Overdispersion leads to underestimating the standard errors of the parameter estimates and overestimates the results of the tests statistics. To correct overdispersion, use the negative binomial distribution in the PROC GENMOD MODEL statement. This distribution also models count data but allows the variance to exceed the mean. Use a STORE statement to save the model for prediction.

```
proc genmod data=sp4r.crab plots=resraw;
  class color(param=ref ref='1') spine(param=ref ref='1');
  model satellites = width weight color spine
    / dist=negbin link=log type3;
  estimate 'my estimate' intercept 1 width 25 weight 2.5
    color 1 0 0 spine 1 0 / e exp alpha=.05;
  output out=sp4r.out p=pred resraw=res;
  store mymod;
run;
```

Selected PROC GENMOD statements:

STORE requests that the procedure save the context and results of the statistical analysis. The resulting item store is a binary file format that cannot be modified.

EXP requests that the exponentiated estimate, its standard error, and its confidence limits be computed. This is desirable when using the log link function.

 The contents of the STORE statement are passed to the PLM procedure for scoring a new data set.

The GENMOD Procedure

Model Information

Data Set	WORK.CRAB
Distribution	Negative Binomial
Link Function	Log
Dependent Variable	Satellites

Number of Observations Read	173
Number of Observations Used	173

Class Level Information

Class	Value	Design Variables		
Color	1	0	0	0
	2	1	0	0
	3	0	1	0
	4	0	0	1
Spine	1	0	0	
	2	1	0	
	3	0	1	

Parameter Information

Parameter	Effect	Color	Spine
Prm1	Intercept		

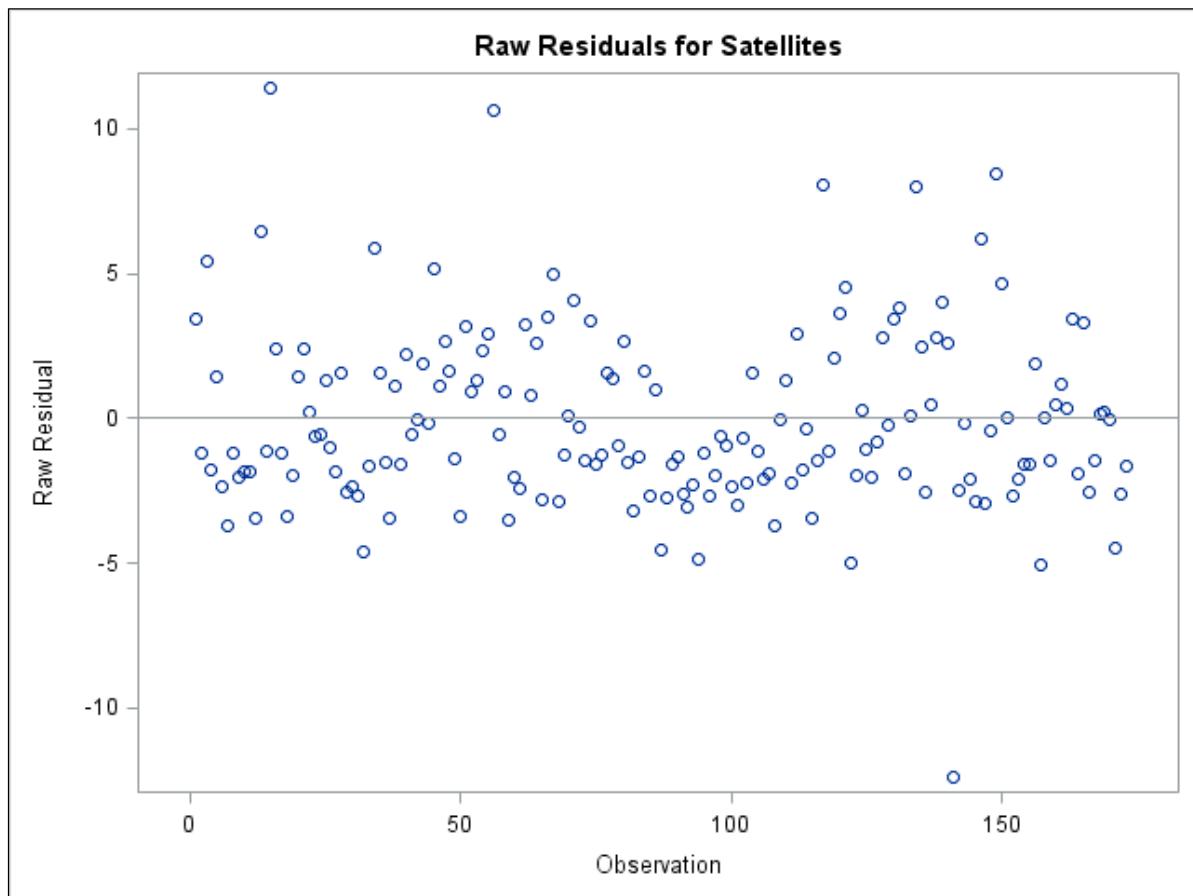
Prm2	Width		
Prm3	Weight		
Prm4	Color 2		
Prm5	Color 3		
Prm6	Color 4		
Prm7	Spine 2		
Prm8	Spine 3		
Criteria For Assessing Goodness Of Fit			
Criterion	DF	Value	Value/DF
Deviance	165	196.5122	1.1910
Scaled Deviance	165	196.5122	1.1910
Pearson Chi-Square	165	154.2446	0.9348
Scaled Pearson X2	165	154.2446	0.9348
Log Likelihood		157.3742	
Full Log Likelihood		-372.6602	

Criteria For Assessing Goodness Of Fit						
Criterion	DF	Value	Value/DF			
AIC (smaller is better)		763.3204				
AICC (smaller is better)		764.4247				
BIC (smaller is better)		791.7000				
Algorithm converged.						
Analysis Of Maximum Likelihood Parameter Estimates						
Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits	Chi-Square	Pr > ChiSq
Intercept	1	-0.2773	2.1254	-4.4430 3.8883	0.02	0.8962
Width	1	-0.0024	0.1108	-0.2197 0.2148	0.00	0.9825
Weight	1	0.7009	0.4038	-0.0906 1.4923	3.01	0.0826
Color	2	1 -0.3208	0.3821	-1.0696 0.4281	0.70	0.4012
Color	3	1 -0.5963	0.4198	-1.4190 0.2265	2.02	0.1555
Color	4	1 -0.5790	0.4782	-1.5162 0.3582	1.47	0.2260
Spine	2	1 -0.2427	0.3894	-1.0058 0.5204	0.39	0.5331
Spine	3	1 0.0428	0.2469	-0.4412 0.5267	0.03	0.8625
Dispersion	1	1.0363	0.1891	0.7247 1.4819		
NOTE: The negative binomial dispersion parameter was estimated by maximum likelihood.						
LR Statistics For Type 3 Analysis						
Source	DF	Chi-Square	Pr > ChiSq			
Width	1	0.00	0.9825			
Weight	1	3.02	0.0823			
Color	3	2.76	0.4297			
Spine	2	0.59	0.7452			

Coefficients for Contrast my estimate	
Label	Prm1 Prm2 Prm3 Prm4 Prm5 Prm6 Prm7 Prm8
my estimate	1 25 2.5 1 0 0 1 0

Contrast Estimate Results							
Label	Mean Estimate	Mean Confidence Limits	L'Beta Estimate	Standard Error	Alpha	L'Beta	Confidence Limits
my estimate	2.3408	1.1626 4.7130	0.8505	0.3571	0.05	0.1507	1.5503
Exp(my estimate)			2.3408	0.8358	0.05	1.1626	4.7130

Contrast Estimate Results		
Label	Chi-Square	Pr > ChiSq
my estimate	5.67	0.0172
Exp(my estimate)		



Comparing the fit statistics for this negative binomial model to the fit statistics for the Poisson model, you can see that the negative binomial model fits the data better.

MODELS		
FIT STATISTICS	Original Model	Negative Binomial Model
AIC (smaller is better)	920.883	763.3204
AICC (smaller is better)	921.7613	764.4247
BIC (smaller is better)	946.1096	791.7000

4. The previous demonstration used a STORE statement to save the estimated model. Use the model to predict the number of satellites when **Color** is 2, **Spine** is 2, **Width** is 25, and **Weight** is 2.5. Create a new data table of the predictor values and pass it to PROC PLM for prediction. Remember to use the ILINK option to predict on the original data scale. Print both the predictor and predicted values.

```
data sp4r.newcrab;
  input Color Spine Width Satellites Weight;
  datalines;
2 2 25 0 2.5
;run;

proc plm restore=mymod;
  score data=sp4r.newcrab out=sp4r.scores / ilink;
run;

proc print data=sp4r.scores;
run;
```

The PLM Procedure	
Store Information	
Item	Store
Data Set Created From	WORK.CRABPRED
Created By	WORK.CRAB
Date Created	PROC Genmod
Response Variable	25SEP15:16:55:31
Link Function	Satellites
Distribution	Log
Class Variables	Negative Binomial
Model Effects	Color Spine
	Intercept Width Weight Color Spine

Obs	Color	Spine	Width	Satellites	Weight	Predicted
1	2	2	25	0	2.5	2.34081

End of Demonstration

6.3 Mixed Models

Objectives

Use PROC MIXED to create the following models:

- randomized complete block design mixed models
- two-way interaction mixed models

78

Motivation

Use a PROC step to perform the following tasks:

- create linear mixed models
- estimate variance components
- test fixed effects and random effects for significance

$$Y = X\beta + Z\gamma + \varepsilon$$



79

Duplicating the R Script

```

Source on Save | Run | Source
#Install the lme4 package
#to compute linear mixed effects models
install.packages("lme4")
library(lme4)

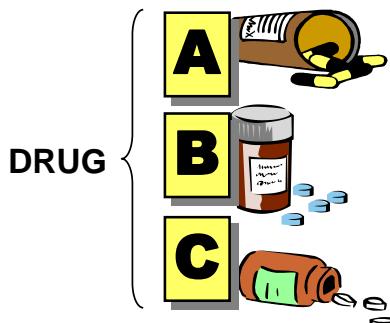
#Create a RCBD model with the LMER function
lmem = lmer(pressure ~ adhesive + (1|toy), data=toys)
summary(lmem)

```

80

Fixed Effects

- All levels of interest are selected by a nonrandom process and are included in the study.
- Inferences are to be made only to those levels included in the study.



81

Fixed effects are those factors whose levels are selected deliberately to evaluate the differences. All levels of interest are in your data set. The researcher is interested in comparing the effects of the factors on the response variable only for those levels included in the study.

For example, in a drug study, you want to compare the effect of three drugs (A, B, and C). You are interested in the comparison of only these three drugs. You know what they are before conducting the experiment.

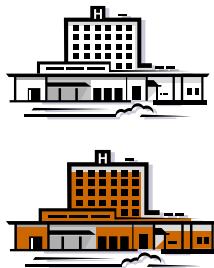
The variable **Drug** is a fixed effect. Other examples of fixed effects that represent all the levels of interest for the study might include **Gender**, **Treatment**, **Method**, and **Brand**.

A model containing only fixed effects is called a *fixed effects model*.

Mixed Models

Models in which some factors are fixed effects and other factors are random effects are called *mixed models*.

CLINIC – random



DRUG – fixed



82

In some situations, a factor might have a large number of levels and the researcher or data analyst selects a subset of the levels to be included in the study. They represent a sample (although often an imperfect sample) from a population with a probability distribution. The inference about fixed effects from the data analysis applies to all population levels of random effects and not only the subset of levels included in the study. Effects such as these are *random effects*. For example, in the same drug study, four clinics are randomly selected from a population of clinics in a region. The researcher wants to make an inference for the drug effects across the population of clinics, not only the ones included in the study. Then **Clinic** is a random effect.

MIXED Procedure Model

$$Y = X\beta + Z\gamma + \varepsilon$$

Random effects

Design matrix
for random effects

No longer required
to be independent
and homogeneous

Assume $\gamma \sim N(0, \mathbf{G})$ and $\varepsilon \sim N(0, \mathbf{R})$

$$\rightarrow E(y) = \mathbf{X}\beta, \quad \text{Var}(y) = \mathbf{Z}\mathbf{G}\mathbf{Z}' + \mathbf{R} = \mathbf{V}$$

83

- y is the vector of observed response data values.
- \mathbf{X} is the known design matrix for the fixed effects resulting from the MODEL statement in PROC MIXED.
- β is the vector of unknown fixed-effect parameters.
- \mathbf{Z} is the known design matrix for the random effects resulting from the RANDOM statement in PROC MIXED.
- γ is the vector of unknown random effects parameters.
- ε is the vector of random errors.

Assume that γ and ε are independently and normally distributed with

$$E\begin{bmatrix} \gamma \\ \varepsilon \end{bmatrix} = 0 \text{ and } Var\begin{bmatrix} \gamma \\ \varepsilon \end{bmatrix} = \begin{bmatrix} G & 0 \\ 0 & R \end{bmatrix}.$$

Estimation Methods in PROC MIXED

- For the covariance parameters
 - Method of Moments
 - MIVQUE0
 - Type 1
 - Type 2
 - Type 3
 - Likelihood-based Methods
 - ML
 - REML (default)
- For the fixed-effects parameters and standard errors
 - generalized least squares (GLS) method

84

The following summarizes the estimation methods provided by PROC MIXED:

MIVQUE0	Performs minimum variance quadratic unbiased estimation of the covariance parameters. It produces method-of-moments estimates that are invariant with respect to the fixed effects. That is, the mean squares associated with the random effects are adjusted for the fixed effects.
Type 1 Type 2 Type 3	Applies only to variance component models with no SUBJECT= effects and no REPEATED statement. An analysis of variance table is included in the output, and the expected mean squares are used to estimate the variance components. These are method-of-moments estimates.
Maximum Likelihood (ML)	Iteratively searches over the parameter space for parameter values that maximize a function of the parameters related to the chance of observing

the data collected. The function being maximized is called the *likelihood function*.

Restricted/Residual Maximum Likelihood (REML)

constructs the likelihood function based on the residuals and obtains maximum likelihood estimates of the variance components from this restricted or residual likelihood function.

Many studies show that the default estimation method of the covariance parameters, REML, has many advantages over other methods. This method is recommended and is used throughout this course. A detailed discussion about the estimation methods of the covariance parameters is provided later in this chapter.

-  MIVQUE0 estimates are used as starting values for REML estimates.

MIXED Procedure

General form of the MIXED procedure:

```
PROC MIXED DATA=data-table-name;
  CLASS variables;
  MODEL dependent-variable = fixed-effects / solution;
  RANDOM random-effects / <options>;
  ESTIMATE 'label' fixed-effect-values
            | random-effect-values / <options>;
  LSMEANS fixed-effects / options;
  RUN;
```

$$Y = X\beta + Z\gamma + \varepsilon$$

85

Selected PROC MIXED statement:

RANDOM specifies the random effects for the model.

-  Random effects are specified only in the RANDOM statement. They are not included in the MODEL statement.

Agriculture Example

Three seed-growth methods are applied to seeds from each of five varieties of turf grass. Six pots are planted with seeds from each method-by-variety combination. The 90 pots are randomly placed in a uniform growth chamber, and dry matter yields are measured from clippings at the end of four weeks.

Assume that the five varieties were randomly chosen from a broader population of varieties.

- two-way mixed model:

$$y_{ijk} = \mu + \alpha_i + b_j + (\alpha b)_{ij} + \varepsilon_{ijk}$$

- variance components:

$$(\alpha b)_{ij} \sim N(0, \sigma_{ab}^2) \quad b_j \sim N(0, \sigma_b^2) \quad \varepsilon_{ijk} \sim N(0, \sigma^2)$$



Two-Way Mixed Model

SP4R06d09.sas

Three seed growth methods are applied to seeds from each of five varieties of turf grass. Six pots are planted with seeds from each method-by-variety combination. The 90 pots are randomly placed in a uniform growth chamber, and dry-matter yields are measured from clippings at the end of four weeks.

This is an example of a completely randomized experiment with a factorial arrangement of treatments. The fifteen treatments are the combinations of levels of the two factors, **variety** (five levels) and **method** (three levels). Assume that the five varieties were randomly chosen from a broader population of varieties. Interest is not in these particular five varieties but in the population from which they were chosen. The variable **variety** is considered to be a *random* effect. The variable **method** is considered to be a *fixed* effect because the interest is only on these three methods. The **method*variety** interaction is a random effect.

Because both random and fixed effects are involved, the model is defined as being mixed. The varieties are random, and the inference about method means that differences should apply across all varieties.

The data are recorded in the data set **Grass**. The following variables are in the data set:

Method the growth method (A, B, or C)

Variety the variety of turf grass (1, 2, 3, 4, or 5)

Yield the amount of dried turf grass in each pot after four weeks

Use PROC MIXED to fit the following model to the data:

$$y_{ijk} = \mu + \alpha_i + b_j + (\alpha b)_{ij} + \varepsilon_{ijk}$$

where

y_{ijk} k^{th} observation ($k=1$ to 6) for the i^{th} method ($i=1, 2, 3$) and the j^{th} variety ($j=1$ to 5)

μ overall mean and an unknown fixed parameter

α_i effect for the i^{th} method and also an unknown fixed parameter

b_j effect of the j^{th} variety, a random effect, $b_j \sim \text{i.i.d. } N(0, \sigma_b^2)$

$(\alpha b)_{ij}$ interaction between the i^{th} method and the j^{th} variety, a random effect $(\alpha b)_{ij} \sim \text{i.i.d. } N(0, \sigma_{ab}^2)$

ε_{ijk} experimental error, $\varepsilon_{ijk} \sim \text{i.i.d. } N(0, \sigma^2)$.

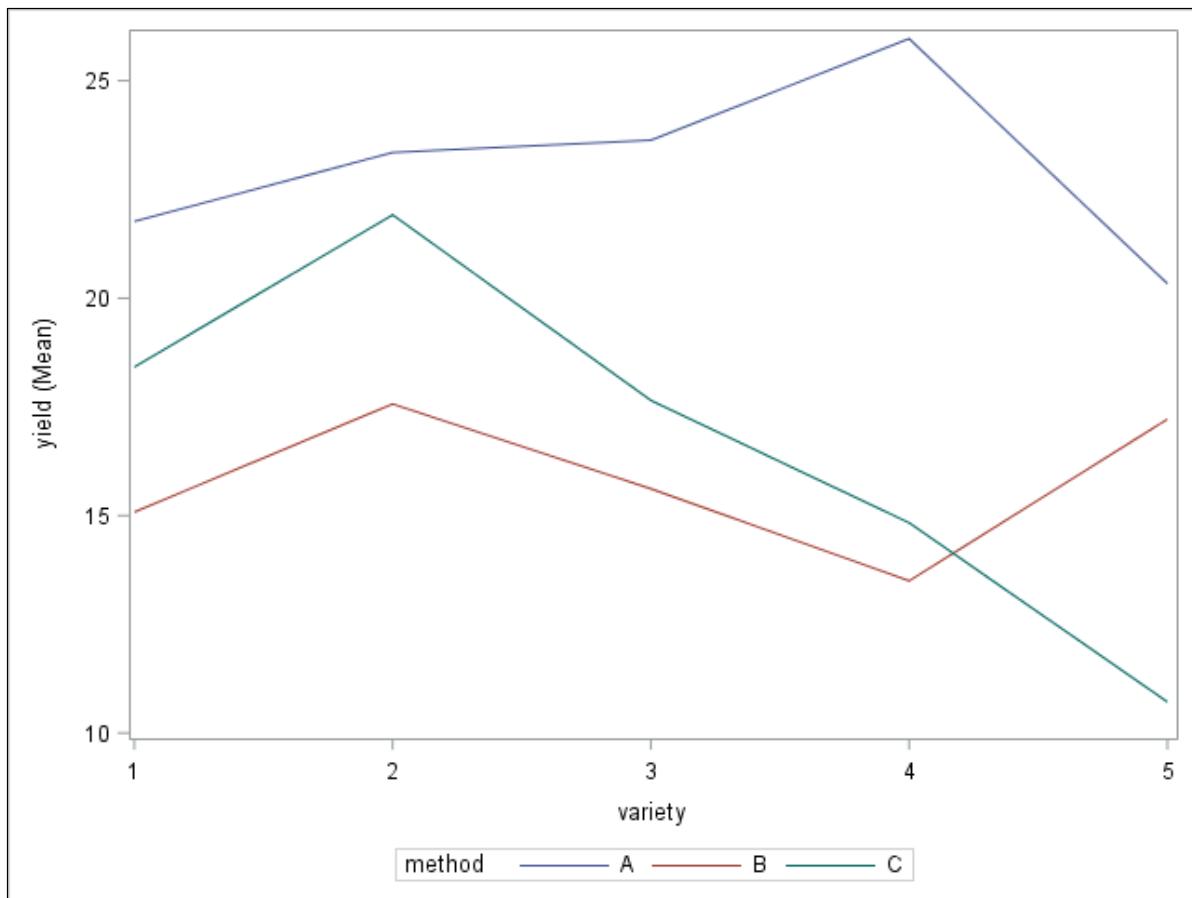
The effects b_j , $(\alpha b)_{ij}$, and ε_{ijk} are assumed to be independent random variables.

Therefore,

- $E(y_{ijk}) = \mu + \alpha_i$ is the mean for method i averaged across all varieties in the population.
- $Var(y_{ijk}) = \sigma_b^2 + \sigma_{ab}^2 + \sigma^2$ is the variance of an observation. The variance components are σ_b^2 (**variety** variance), σ_{ab}^2 (**method*variety** variance), and σ^2 (random errors).

1. Use PROC SGPlot to explore the data.

```
proc sgplot data=sp4r.grass;
    vline variety / group=method stat=mean response=yield;
run;
```



There seems to be some variability among varieties. In addition, the yield for Method A is largest for all five varieties.

2. Use PROC MIXED to create a two-way mixed model and use the METHOD=REML option. (This method reproduces the R package *lmer*.) Remember that the random effects appear only in the RANDOM statement, not the MODEL statement. However, all classification variables, fixed and random effects, are listed in the CLASS statement. Use an LSMEANS statement to compute the least square means for **Method** and use the PDIFF option to evaluate the difference in methods. Finally, use an ESTIMATE statement to compare Method A versus B and C.

```
proc mixed data=sp4r.grass method=REML;
  class method variety;
  model yield = method / solution ddfm=kr2;
  random variety method*variety;
  lsmeans method / pdiff;
  estimate 'A vs. B and C' method 1 -.5 -.5;
run;
```

The Mixed Procedure

Model Information

Data Set	WORK.GRASS
Dependent Variable	yield
Covariance Structure	Variance Components
Estimation Method	REML

Residual Variance Method	Profile
Fixed Effects SE Method	Kenward-Roger2
Degrees of Freedom Method	Kenward-Roger2

The Model Information table describes the model, some of the variables that it involves, and the method that is used to fit it. The default covariance structure fitted by PROC MIXED is variance components (VC) (constant variances and zero covariance), also referred to as *independent covariance structure*. Restricted or residual maximum likelihood (REML) is the default variance component estimation method used by PROC MIXED.

There are four methods for handling the residual variance in the model. The *profile* method concentrates the residual variance out of the optimization problem. This means solving analytically for the optimal residual variance and plugging this expression back into the likelihood formula. This reduces the number of optimization parameters by one and can improve convergence properties. The *fit* method retains it as a parameter in the optimization. The *factor* method keeps the residual fixed, and *none* is displayed when a residual variance is not a part of the model.

The row labeled *Fixed Effects SE Method* in this table describes the method used to compute the approximate standard errors for the fixed-effects parameter estimates and related functions of them. The default method is model-based, which can be changed to empirical when the EMPIRICAL option in the PROC MIXED statement is used, or Prasad-Rao-Jeske-Kackar-Harville when the DDFM=KR2 option is specified in the MODEL statement.

- ☞ DDFM=KR2 applies the (prediction) standard error and degrees-of-freedom correction that are detailed by Kenward and Roger (2009). This correction reduces the bias of the precision estimator for fixed effects under nonlinear covariance structures.

The row labeled *Degrees of Freedom Method* in this table lists the method used for estimating the denominator degrees of freedom for the fixed effect. Seven possibilities for this are Containment, Between-Within, Residual, Satterthwaite, Kenward-Roger, Kenward-Roger (Firstorder) and Kenward-Roger2. Containment is the default method when the RANDOM statement is specified. You can use the DDFM= option in the MODEL statement to specify other methods.

Class Level Information		
Class	Levels	Values
method	3	A B C
variety	5	1 2 3 4 5
Dimensions		
Covariance Parameters		3
Columns in X		4
Columns in Z		20
Subjects		1
Max Obs per Subject		90
Number of Observations		
Number of Observations Read		90
Number of Observations Used		90
Number of Observations Not Used		0

The Dimensions table lists the sizes of relevant matrices. Remember that the X matrix refers to the fixed effects design matrix and the Z matrix refers to the random effects design matrix. The X design matrix includes the intercept term (4 = intercept + 3 levels of **Method**). The covariance parameters includes the parameters from the RANDOM statement and the error term.

Iteration History					
Iteration	Evaluations	-2 Res Log Like	Criterion		
0	1	528.89057283			
1	1	522.49142693	0.00000000		
Convergence criteria met.					

The Iteration History table describes the optimization of the residual log likelihood function. The optimization is performed using a ridge-stabilized Newton-Raphson algorithm, and the rows of this table describe the iterations that this algorithm takes in order to minimize the objective function.

Covariance Parameter Estimates	
Cov Parm	Estimate
variety	0.4285
method*variety	4.7715
Residual	18.4347

The variance component estimate are $\hat{\sigma}_b^2 = 0.4285$, $\hat{\sigma}_{ab}^2 = 4.7715$, and $\hat{\sigma}^2 = 18.4347$.

Fit Statistics	
-2 Res Log Likelihood	522.5
AIC (Smaller is Better)	528.5
AICC (Smaller is Better)	528.8
BIC (Smaller is Better)	527.3

The Fit Statistics table provides information for goodness of model fit. All three information criteria consider both the fitness of the model and the model complexity. The model with more parameters receives a larger penalty. Bayesian information criterion (BIC) tends to produce a larger penalty than both Akaike information criterion (AIC) and finite-sample corrected Akaike information criterion (AICC). The detailed formula for the computations of these information criteria can be found in the SAS online documentation. For all three information criteria, smaller values indicate a better model.

Solution for Fixed Effects						
Effect	method	Estimate	Standard	DF	t Value	Pr > t
			Error			
Intercept		16.7067	1.2863	11.9	12.99	<.0001
method	A	6.3033	1.7713	8	3.56	0.0074
method	B	-0.9100	1.7713	8	-0.51	0.6213
method	C	0

Type 3 Coefficients for method			
Effect	method	Row1	Row2
	Intercept		

method	A	1		
method	B		1	
method	C	-1	-1	
Type 3 Tests of Fixed Effects				
Effect	Num DF	Den DF	F Value	Pr > F
method	2	8	9.84	0.0070

The Type 3 Coefficients for **Method** print the Type 3 **L** matrix that is used for the test for **Method** when the E option is used in the MODEL statement.

The Type 3 Tests of Fixed Effects table contains the F test for **adhesive** ($F=9.84$), with a p -value of 0.0070. Therefore, you conclude that there is a difference between the yield for the three methods at a 5% significance level.

 PROC MIXED does not compute the sums of squares as PROC GLM does. Therefore, you do not see sums of squares in the Test of Fixed Effects table. The F statistic for the fixed effect is

$$\text{computed as } F = \frac{\hat{\beta}' L' [L(X'\hat{V}^{-1}X)^{-1}L]^{-1} L \hat{\beta}}{\text{rank}(L)}, \text{ where}$$

$\hat{\beta}$ is the vector of the fixed-effect estimates.

L is the coefficient matrix discussed earlier.

X is the design matrix for the fixed effects.

\hat{V} is the estimated covariance matrix for an observation using the REML method.

This statistic accounts for all variance components in the model as indicated by \hat{V} in the equation.

Estimates						
Label		Estimate	Standard Error	DF	t Value	Pr > t
A vs. B and C		6.7583	1.5340	8	4.41	0.0023
Least Squares Means						
Effect	method	Estimate	Standard Error	DF	t Value	Pr > t
method	A	23.0100	1.2863	11.9	17.89	<.0001
method	B	15.7967	1.2863	11.9	12.28	<.0001
method	C	16.7067	1.2863	11.9	12.99	<.0001
Differences of Least Squares Means						
Effect	method	_method	Estimate	Standard Error	DF	t Value
method	A	B	7.2133	1.7713	8	4.07
method	A	C	6.3033	1.7713	8	3.56
method	B	C	-0.9100	1.7713	8	-0.51
						0.6213

The ESTIMATE and LSMEANS statements provide the same output as the previous procedures in the chapter. Here, the methods account for multiple variance components.

3. Conduct the same analysis except change the METHOD= option in the PROC MIXED statement to TYPE3. To suppress duplicate output, request only the Type3 table.

```
ods select type3;
proc mixed data=sp4r.grass method=type3;
  class method variety;
  model yield = method / solution ddfm=kr2;
  random variety method*variety;
  lsmeans method / pdiff;
  estimate 'A vs. B and C' method 1 -.5 -.5;
run;
```

The Mixed Procedure				
Type 3 Analysis of Variance				
Source	DF	Sum of Squares	Mean Square	Expected Mean Square
method	2	925.922889	462.961444	Var(Residual) + 6 Var(method*variety) + Q(method)
variety	4	219.104889	54.776222	Var(Residual) + 6 Var(method*variety) + 18 Var(variety)
method*variety	8	376.510444	47.063806	Var(Residual) + 6 Var(method*variety)
Residual	75	1382.600000	18.434667	Var(Residual)
Type 3 Analysis of Variance				
Source	Error Term		Error	
	DF	F Value	Pr > F	
method	MS(method*variety)	8	9.84	0.0070
variety	MS(method*variety)	8	1.16	0.3946
method*variety	MS(Residual)	75	2.55	0.0162
Residual

The METHOD=TYPE3 option in the PROC MIXED statement produces the sum of squares table that is similar to the GLM procedure. In addition, hypothesis tests for variance components are tested according the expected sum of squares. The Error Term column indicates which mean square is used as the denominator in the test of a specified variance component. For example, the Variety variance component is tested using the Method*Variety mean square interaction.

End of Demonstration

6.4 Other Procedures

Objectives

List and describe the procedures that you can use for the following statistical analyses:

- generalized linear mixed models
- Bayesian
- survival
- multivariate
- time series



89

Generalized Linear Mixed Models

PROC MIXED: $Y = X\beta + Z\gamma + \varepsilon$

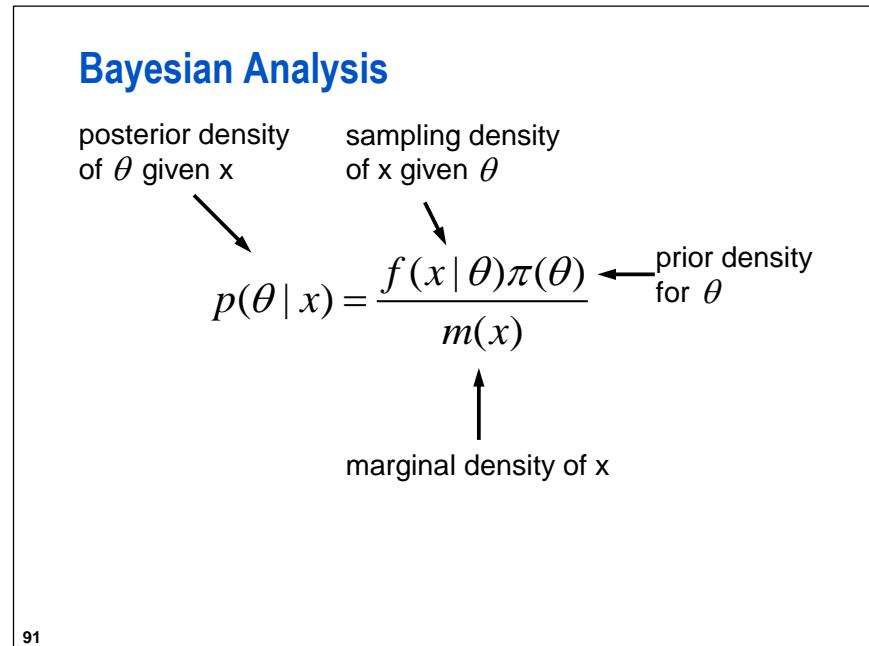
PROC GLIMMIX: $g(\mu) = X\beta + Z\gamma$

```
PROC GLIMMIX DATA=data-table-name;
  CLASS variables;
  MODEL dependent-variable = fixed-effects / SOLUTION
    DIST=probability-distribution LINK=link-function;
    RANDOM random-effects / <options>;
    ESTIMATE 'label' fixed-effect-values | random-effect-values
      / <options>;
    LSMEANS fixed-effects / options;
RUN;
```

90

In the generalized linear mixed model, you apply a LINK function to the conditional mean $E(y|\gamma)$. The conditional distribution of $y|\gamma$ plays the same role as the distribution of y in the fixed-effects generalized linear model. You apply the same basic strategies for fitting a generalized linear model to $E(y)$ in a fixed effect model to fitting a mixed model to conditional mean $E(y|\gamma)$.

However, to obtain the parameter estimates, you must obtain the marginal log-likelihood function, which is a challenge when you fit generalized linear mixed models (and nonlinear mixed models). By default, the GLIMMIX procedure uses the linearization technique to approximate the generalized linear mixed model as a linear mixed model. Two maximum likelihood methods are also available for fitting generalized linear mixed models in PROC GLIMMIX.



91

The above equation is often expressed as follows:

$$\text{posterior density} = (\text{likelihood} * \text{prior}) / \text{marginal likelihood}$$

The marginal density of x is an integral defined as follows:

$$\int f(x | \theta)\pi(\theta)d\theta$$

The posterior density or distribution describes the distribution of the parameter of interest with respect to the data and prior. The posterior distribution is necessary for probabilistic prediction and for sequential updating.

Although the name *prior* suggests a temporal relationship, it is feasible for a prior distribution to be decided after seeing the results of the study (for example, empirical Bayes methods). Cox (1999) points out that the prior distribution refers to a situation where you assess what the evidence would be if you had no data. This assessment can be made after seeing the data, but there are issues in this.

There is no such thing as the “correct” prior. In fact, researchers, such as Kass and Greenhouse (1989), suggested using a “community of priors” to describe the range of reasonable opinions.

Even though Bayesian analysis is driven by the prior distribution, it is sometimes not important in the analysis. As the sample size increases, the prior usually is overwhelmed by the likelihood and exerts a negligible influence on the conclusions. However, Bayesian analysis is not based on this assumption.

The development of the posterior distribution might be difficult. The specific problem is carrying out the integrations that are necessary to obtain the posterior distributions of quantities of interest in situations where nonstandard prior distributions are used. For many years, these problems in integration restricted Bayesian applications to rather simple examples involving conjugate priors.

Most Bayesian analyses require sophisticated computations, including the use of simulation methods such as the Monte Carlo methods, to generate samples from the posterior distribution. The basic idea of Monte Carlo is to simulate the sampling process from a defined population repeatedly by using a computer instead of actually drawing multiple samples to estimate the population summaries of the events of interest.

Markov Chain Monte Carlo methods (MCMC) enable researchers to directly sample sequences of values from the posterior distribution of interest, foregoing the need for closed form analytic solutions. With MCMC, you use these samples to estimate the posterior distribution's quantities of interest. MCMC methods sample successively from a target distribution. Each sample depends on the previous one, hence, the notion of the Markov chain. You can think of a Markov chain applied to sampling as a mechanism that traverses randomly through a target distribution without having any memory of where it was given the immediate past value. Where it moves next is entirely dependent on where it is now.

The Markov chain method is quite successful in modern Bayesian computing. One reason is that if the simulation algorithm is implemented correctly, the Markov chain is guaranteed to converge to the target distribution under rather broad conditions, regardless of the initial values of the parameters. Therefore, the Markov chain is able to improve its approximation to the true distribution at each step in the simulation. Furthermore, the simulation algorithm is easily extensible to models with a large number of parameters or high complexity (SAS Institute, Inc. 2010).

Bayesian Analysis – PROC MCMC

$$Y_i \sim \text{normal}(\beta_0 + \beta_1 X_i, \sigma^2)$$

for subjects $i=1,2,\dots,n$

```
proc mcmc data=slr nbi=2000 nmc=10000;
  parms beta0 0 betal 0;
  parms sigma2 1;
  prior beta0 betal ~ normal(mean=0,
    var=1e6);
  prior sigma2 ~ igamma(shape=2.001,
    scale=1.001);
  mu=beta0 + betal*X1;
  model Y ~ normal(mu, var=sigma2);
run;
```

92

PROC MCMC uses a random walk Metropolis algorithm to obtain posterior samples.

Selected PROC MCMC statements and options:

NBI specifies the number of burn-in iterations to perform before beginning to save parameter estimate chains. By default, NBI=1000.

NMC specifies the number of iterations in the main simulation loop. By default, NMC=1000.

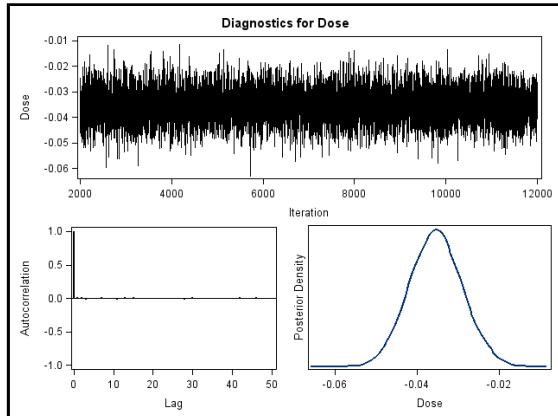
PARMS lists the names of the parameters in the model and specifies option initial values.

PRIOR specifies the prior distribution for each model parameter.

MODEL specifies the conditional distribution of the data given the parameters (the LIKELIHOOD function). You must specify a single dependent variable, a tilde, and then a distribution with its arguments.

Bayesian Analysis – PROC MCMC

$$Y_i \sim \text{normal}(\beta_0 + \beta_1 X_i, \sigma^2)$$

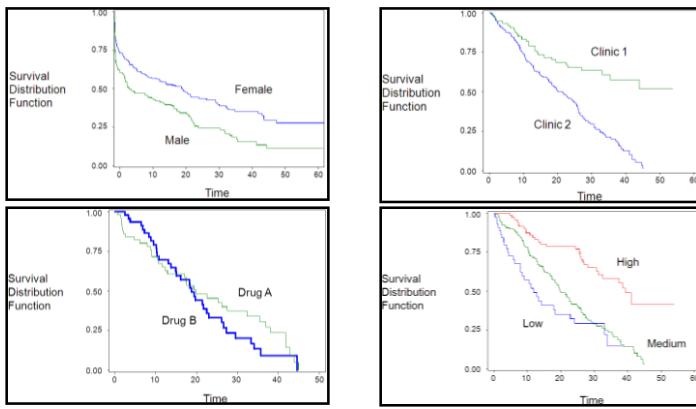


93

PROC MCMC computes a trace plot and density estimate for each model parameter.

Survival Analysis – PROC LIFETEST

The LIFETEST procedure computes and plots survival functions and tests for differences between survival functions.



94

Survival analysis is a collection of specialized methods that are used to analyze data in which time until an event occurs is the response variable of interest. The response variable (often called, in survival analysis, a *failure time*, *survival time*, or *event time*) is usually continuous and can be measured in days, weeks, months, years, and so on. Events can be deaths, onset of disease, marriages, arrests, and so on. What is unique about survival analysis is that even if the subject did not experience an event, the subject's survival time or length of time in the study is taken into account.

Survival analysis is used heavily in clinical and epidemiological follow-up studies. Other fields that use survival analysis methods include sociology, engineering, and economics. Survival analysis is also known as time-to-event analysis, reliability analysis, durability analysis, event history analysis, and lifetime analysis, among others. Regardless of the field, the common objective of a survival analysis study is not only ***whether*** an event occurred, but also ***when*** it occurred. For example, subjects who die five years after surgery are different from subjects who die one month after surgery. An analysis that simply counted deaths ignores valuable information about survival time.

Survival analysis can also be used to analyze outcomes other than time. For example, an engineer might want to analyze the amount of mileage until a tire fails or the number of cycles until an engine requires repair. What is common across these studies is that you are analyzing an outcome until an event occurs, and that outcome does not necessarily have to be time.

Survival analysis allows the response variable to be incompletely determined for some subjects. Exact failure time remains unknown. When this occurs, it is called *censoring*. These subjects should not be ignored. The time at which they are observed contributes information to the study. Ignoring them completely adds bias to the estimates of population survival time. They should not be assumed to have the event at the closest observed time point because event times (assuming an event eventually occurs) reported that way would be inaccurately measured.

Censoring is categorized into three main types: *right*, *left*, and *interval*, depending on where the lack of information exists on the timeline relative to the observed follow-up times.

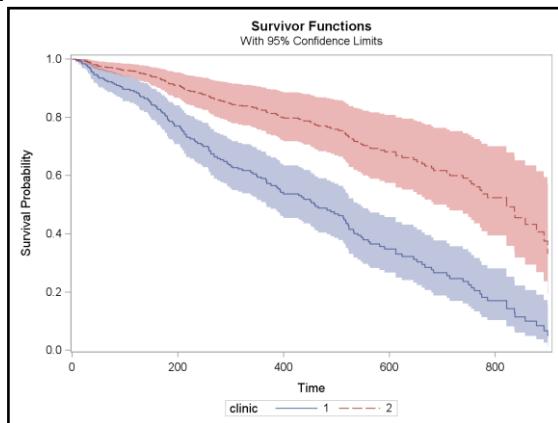
Usually, the first step in the analysis of survival data is to estimate and plot the survival function. The survival function gives the probability that a subject survives longer than some specified time t . This can be defined by the formula $S(t) = \Pr(T > t)$ where T is a random variable for a person's survival time and t is any specific value of interest. At $t=0$, $S(0)=1$ (at the start of the study, because no one experienced the event yet, the probability of surviving past time 0 is 1), while at $t=\infty$, $S(\infty)=0$ (eventually nobody survives, so the survival function theoretically must fall to 0). As t increases, $S(t)$ never increases and usually decreases. The factors that influence the shape of the survival function are when the subjects experience the event, when the subjects were censored, and the pattern of enrollment in the follow-up study (Hosmer and Lemeshow 1999). In practice, the survival function resembles a decreasing step function rather than a smooth curve. Furthermore, because not everyone might experience the event by the end of the study (they are Type I censored), the survival function might not reach 0.

A useful graph in exploratory data analysis is a graph that compares survival functions across groups. In the plot at the upper left, the female survival function lies above the male survival function, which means that females had a more favorable survival experience. If the event was death, then at any point in time, the proportion of females estimated to be alive is larger than the proportion of males estimated to be alive.

A graph comparing survival functions can also give insight into how time is related to the survival experience across groups. It can indicate interactions with time. In the plot at the upper right, subjects in Clinic 1 have a more favorable survival experience than subjects in Clinic 2. However, the differences between the groups are relatively small in the early time points and become progressively larger in the later time points. Early in the study, both clinics lost a similar proportion of patients. However, as the study progressed, the patients in Clinic 1 had much longer survival times compared to the patients in Clinic 2.

Survival Analysis – PROC PHREG

The PHREG procedure performs regression analysis of survival data based on the Cox proportional hazard model.



95

In many situations, either the true form of the hazard function is unknown or it is so complex that the distributions covered in PROC LIFEREG do not adequately describe your data. This is a problem in parametric models because one of the assumptions is that the true form of the underlying hazard function is correctly specified. Therefore, the parameter estimates of the survival model might be biased if the wrong distribution is specified (Harrell 1997).

This problem was addressed in 1972 by the British statistician Sir David Cox in a paper called “Regression Models and Life Tables.” In his paper, Cox proposed a model (now called the Cox proportional hazards model) that does not require that the distribution of survival times be known. It is a semi-parametric model because it makes a parametric assumption concerning the effect of the predictor variables on the hazard function. (It assumes that the predictor variables act multiplicatively on the hazard function.) However, the model makes no assumption regarding the nature of the hazard function. For example, the model does not assume that the hazard function is constant (the exponential model), or that it follows the form specified in a Weibull model or any other parametric model.

The Cox model is extremely popular because, in many instances, the modeling goal of survival data is to characterize how the distribution of survival times changes as a function of the predictor variables. For example, suppose a clinical trial was designed to test whether one drug therapy improves the survival of AIDS patients when compared to another drug therapy. The primary importance of the survival model is to estimate parameters that compare the survival experience of the two treatment groups. The description of the underlying distribution of survival time is not important. Therefore, the actual form of the baseline hazard function is not important.

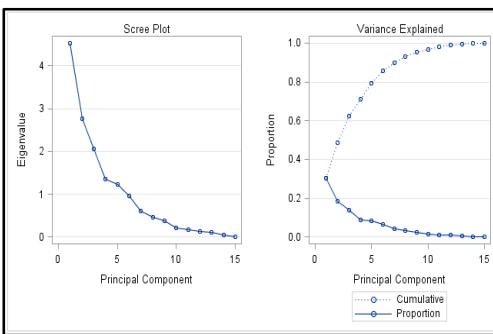
Another reason that the Cox model is popular is because the model is as efficient in estimating and testing regression coefficients as the parametric models even when the distribution is correctly specified. When the distribution of survival times is incorrectly specified, the Cox model is more efficient than the parametric models (Harrell 1997).

The Cox model also uses only the rank ordering of the event and censoring times. This property makes the model less affected by outliers in the event times than in parametric models.

Multivariate Analysis

Examples of common multivariate procedures are as follows:

- PROC PRINCOMP
- PROC FACTOR
- PROC CANCORR
- PROC CANDISC
- PROC DISCRIM
- PROC PLS



96

Multivariate analysis refers to a broad category of statistical methods used when more than one variable at a time is analyzed for a subject. Although many physical and virtual systems studied in scientific and business research are, by their very nature, multivariate (that is, there are many responses influenced by many different variables simultaneously), most analyses are univariate (analyzing only one response at a time) in practice.

Principal components analysis is a dimension reduction technique that creates new variables that are a weighted linear combination of a set of correlated variables.

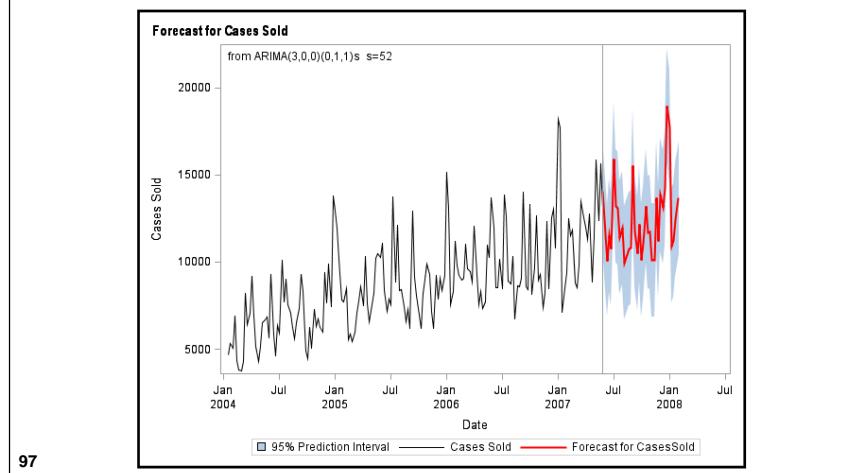
Factor analysis is similar to principal components analysis and is used as a variable identification technique.

Canonical correlation analysis tests hypotheses from multivariate regression but also enables you to interpret how the predictors are related to the responses, interpret how the responses are related to the predictors, and examine how many dimensions the variable sets share in common.

Discriminant function analysis is a method that can be used to identify linear combinations of variables that produce the greatest distance between categories and to classify observations into groups.

Time Series

Model a time series and forecast future values with PROC ARIMA.



The ARIMA procedure analyzes and forecasts equally spaced univariate time series data, transfer function data, and intervention data by using the autoregressive integrated moving average (ARIMA) or autoregressive moving average (ARMA) model. An ARIMA model predicts a value in a response time series as a linear combination of its own past values, past errors (also called shocks or innovations), and current and past values of other time series.

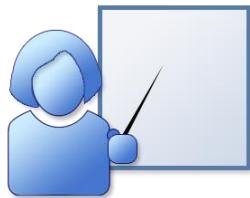
The ARIMA procedure provides a comprehensive set of tools for univariate time series model identification, parameter estimation, and forecasting, and it offers great flexibility in the types of ARIMA or ARIMAX models that can be analyzed. The ARIMA procedure supports seasonal, subset, and factored ARIMA models; intervention or interrupted time series models; multiple regression analysis with ARMA errors; and rational transfer function models of any complexity.

The design of PROC ARIMA closely follows the Box-Jenkins strategy for time series modeling with features for the identification, estimation, and diagnostic checking, and forecasting steps of the Box-Jenkins method.

Conclusion

When in doubt, search for it!

- Consult the online SAS documentation to find the SAS procedure that meets your needs:
<http://support.sas.com/documentation/>
- Watch free SAS videos to learn SAS syntax tips and tricks or take a class to learn new analytic procedures:
<http://support.sas.com/training/>





Exercises

1. Fitting a Regression Model

Percentage of body fat, age, weight, height, and 10 body circumference measurements were recorded for 252 men by Dr. Roger W. Johnson of Calvin College in Minnesota. The data are in the **BodyFat** data set, which consists of the following variables:

Case	Case number
PetBodyFat2	Percent body fat using Siri's equation (495/density - 450)
Age	Age in years
Weight	Weight in pounds
Height	Height in inches
Neck	Neck circumference (cm)
Chest	Chest circumference (cm)
Abdomen	Abdomen circumference (cm)
Hip	Hip circumference (cm)
Thigh	Thigh circumference (cm)
Knee	Knee circumference (cm)
Ankle	Ankle circumference (cm)
Biceps	Extended biceps circumference (cm)
Forearm	Forearm circumference (cm)
Wrist	Wrist circumference (cm)

- a. Generate PROC CORR output for the variables **Height**, **Neck**, **Chest**, and **Weight**. Which variables are highly correlated with **Weight**?

The CORR Procedure						
Simple Statistics						
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
Height	252	70.30754	2.60958	17718	64.00000	77.75000
Neck	252	37.99206	2.43091	9574	31.10000	51.20000
Chest	252	100.82421	8.43048	25408	79.30000	136.20000
Weight	252	178.92440	29.38916	45089	118.50000	363.15000

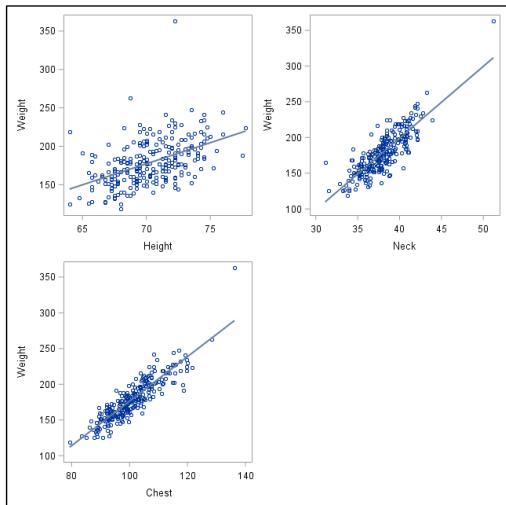
Pearson Correlation Coefficients, N = 252				
Prob > r under H0: Rho=0				
	Height	Neck	Chest	Weight
Height	1.00000	0.32114 <.0001	0.22683 0.0003	0.48689 <.0001
Neck	0.32114 <.0001	1.00000	0.78484 <.0001	0.83072 <.0001
Chest	0.22683 0.0003	0.78484 <.0001	1.00000	0.89419 <.0001
Weight	0.48689 <.0001	0.83072 <.0001	0.89419 <.0001	1.00000

- b. Generate PROC UNIVARIATE output with the same variables. Use the ODS SELECT statement to request only the BasicMeasures table. Do any of the variables appear to be skewed judging only from the mean and median summary statistics?

The UNIVARIATE Procedure			
Variable: Height			
Basic Statistical Measures			
Location		Variability	
Mean	70.30754	Std Deviation	2.60958
Median	70.00000	Variance	6.80992
Mode	71.50000	Range	13.75000
		Interquartile Range	4.00000
The UNIVARIATE Procedure			
Variable: Neck			
Basic Statistical Measures			
Location		Variability	
Mean	37.99206	Std Deviation	2.43091
Median	38.00000	Variance	5.90934
Mode	38.50000	Range	20.10000
		Interquartile Range	3.05000
The UNIVARIATE Procedure			
Variable: Chest			
Basic Statistical Measures			
Location		Variability	
Mean	100.8242	Std Deviation	8.43048
Median	99.6500	Variance	71.07292
Mode	99.1000	Range	56.90000
		Interquartile Range	11.15000

The UNIVARIATE Procedure			
Variable: Weight			
Basic Statistical Measures			
Location		Variability	
Mean	178.9244	Std Deviation	29.38916
Median	176.5000	Variance	863.72272
Mode	152.2500	Range	244.65000
		Interquartile Range	38.25000

- c. Use PROC SGSCATTER to plot **Weight** by **Height**, **Neck**, and **Chest** separately. Add the regression line to each plot. Does each variable appear to be linearly associated with **Weight**?

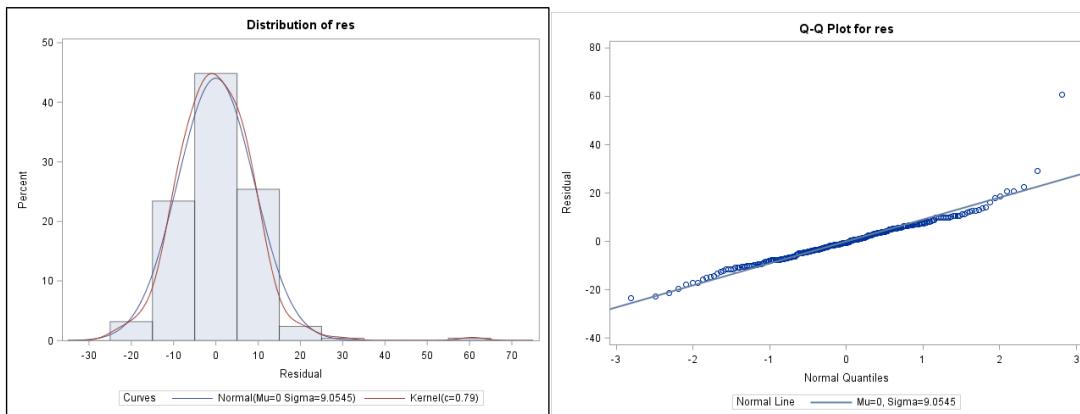


- d. Use PROC REG to create a multiple linear regression model with **Weight** as the dependent variable and **Height**, **Neck**, and **Chest** as independent variables. Use the ODS SELECT statement to request only the tables ANOVA, FitStatistics, and ParameterEstimates. Use the OUTPUT statement to create a new data set with the predicted and residual values. Which variables are statistically significant for this model?

The REG Procedure					
Model: MODEL1					
Dependent Variable: Weight					
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	196216	65405	788.25	<.0001
Error	248	20578	82.97585		
Corrected Total	251	216794			
Fit Statistics					
Root MSE	9.10911	R-Square	0.9051		
Dependent Mean	178.92440	Adj R-Sq	0.9039		
Coeff Var	5.09104				

Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	-366.66231	16.07211	-22.81	<.0001
Height	1	2.96912	0.23287	12.75	<.0001
Neck	1	2.87083	0.39287	7.31	<.0001
Chest	1	2.25905	0.11015	20.51	<.0001

- e. Use PROC UNIVARIATE to create a histogram with a normal density estimate and a Q-Q plot compared to a normal distribution of the residuals. Use the ODS SELECT statement to request only the histogram and Q-Q plot. Do the residuals appear to be normally distributed?



2. Predicting New Data

- Rerun the model from Exercise 1, but this time use a STORE statement to save the model.
- The data set **Newdata_Bodyfat_Reg** contains five new observations. Use PROC PLM to score the new data. Use the PREDICTED keyword in the SCORE statement and save the scored data set as **Pred_Newdata_Bodyfat**.
- Print the predicted values from the scored data set and the response and independent variables.

Obs	Weight	Height	Neck	Chest	Predicted
1	179.00	68.00	39.1	103.3	180.847
2	200.50	69.75	41.3	111.4	210.657
3	140.25	68.25	33.9	86.0	127.579
4	148.75	70.00	35.5	86.7	138.950
5	151.25	67.75	34.5	90.2	137.305

3. Fitting an ANOVA Model

The data set **Cars** contains information about a sample of 1993 model cars from the *1993 Cars Annual Auto Issue* published by *Consumer Reports* and from *Pace New Car and Truck 1993 Buying Guide*. The data set consists of the following variables:

Make	Name of the manufacturer
Model	Name of the model
Type	Vehicle type (Hybrid, SUV, Sedan, Sports, Truck, or Wagon)
Origin	Vehicle origin (Asia, Europe, or USA)

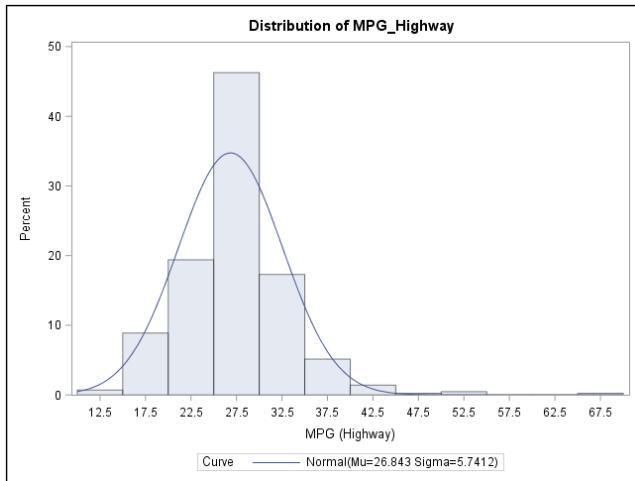
DriveTrain	Drivetrain type (All, Front, or Rear)
Invoice	Invoice price
MSRP	Manufacturer's suggested retail price
EngineSize	Engine displacement size in liters
Cylinders	Number of Cylinders
Horsepower	Maximum horsepower
MPG_City	Average city miles per gallon (EPA rating)
MPG_Highway	Average highway miles per gallon (EPA rating)
Weight	Weight of vehicle in pounds
Wheelbase	Wheelbase in inches
Length	Length of the vehicle in inches

- a. Generate a frequency table for the variable **Type**. Are the counts of each vehicle in this sample evenly distributed?

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	0.70	3	0.70
SUV	60	14.02	63	14.72
Sedan	262	61.21	325	75.93
Sports	49	11.45	374	87.38
Truck	24	5.61	398	92.99
Wagon	30	7.01	428	100.00

- b. Use PROC UNIVARIATE to analyze the **MPG_Highway** variable. Request only the Moments table and the histogram plot with a density estimate. Does **MPG_Highway** appear to be normally distributed?

Moments			
N	428	Sum Weights	428
Mean	26.8434579	Sum Observations	11489
Std Deviation	5.74120072	Variance	32.9613857
Skewness	1.25239527	Kurtosis	6.04561068
Uncorrected SS	322479	Corrected SS	14074.5117
Coeff Variation	21.3877092	Std Error Mean	0.27751141



- c. Use PROC GLM to create a one-way ANOVA model. Use **MPG_Highway** as the dependent variable and **Type** as the independent variable. Use the PLOTS(ONLY)= option in the PROC GLM statement to request only the ANCOVA plot and use the hybrid vehicle as the reference category. Use the SOLUTION and CLPARM options in the MODEL statement to view parameter estimates and confidence limits. Identify the least squares means (LS-means). Use the ADJUST=TUKEY option and the PDIFF and CL options in the LSMEANS statement. Finally, use the ESTIMATE statement to see whether there is a significant difference in **MPG_Highway** for **SUV** versus **Truck**. Are all parameter estimates statistically significant? Are **SUV** and **Truck** significantly different?

Partial PROC GLM Output

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	5	6743.47900	1348.69580	77.64	<.0001
Error	422	7331.03268	17.37212		
Corrected Total	427	14074.51168			

Parameter	Estimate	Standard			
		Error	t Value	Pr > t	95% Confidence Limits
SUV vs Truck	-0.50000000	1.00666449	-0.50	0.6197	-2.47870110 1.47870110

4. Fitting an ANCOVA Model

- a. Extend the ANOVA model above to an ANCOVA model by adding the **HorsePower** variable into the set of independent variables. First, create a macro variable of the mean value of the **HorsePower** variable.

215.8855

- b. Use PROC GLM to fit an ANCOVA model. Use the PLOTS(ONLY)= option to request only the ANCOVA plot from the GLM procedure. Identify the LS-means for the variable **Type**. Use the AT option to hold the variable **HorsePower** fixed at the mean and use the ADJUST=TUKEY option. Estimate the significance of the same linear combination as before, **SUV** versus **Truck**. Are all parameter estimates statistically significant? Is **SUV** still significantly different from **Truck**?

Partial PROC GLM Output

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	10765.94182	978.72198	123.06	<.0001
Error	416	3308.56986	7.95329		
Corrected Total	427	14074.51168			

Parameter	Estimate	Standard			
		Error	t Value	Pr > t	95% Confidence Limits
SUV vs Truck	-0.09692078	0.69865201	-0.14	0.8897	-1.47024909 1.27640754

5. Fitting a Stepwise ANOVA Model

- a. Use the **Cars** data set to practice interacting with the GLMSELECT procedure. Create frequency tables for the variables **Type**, **Origin**, and **DriveTrain**.

The FREQ Procedure					
Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent	
Hybrid	3	0.70	3	0.70	
SUV	60	14.02	63	14.72	
Sedan	262	61.21	325	75.93	
Sports	49	11.45	374	87.38	
Truck	24	5.61	398	92.99	
Wagon	30	7.01	428	100.00	
Origin	Frequency	Percent	Cumulative Frequency	Cumulative Percent	
Asia	158	36.92	158	36.92	
Europe	123	28.74	281	65.65	
USA	147	34.35	428	100.00	
Drive Train	Frequency	Percent	Cumulative Frequency	Cumulative Percent	
All	92	21.50	92	21.50	
Front	226	52.80	318	74.30	
Rear	110	25.70	428	100.00	

- b. Create a model with **MPG_Highway** as the dependent variable and the independent variables **Type**, **Origin**, and **DriveTrain** with all possible interactions. Specify forward selection and let the significance level to entry be 0.05. In addition, use the HIERARCHY=SINGLE option in the MODEL statement to enforce hierarchy. Which parameters were chosen for the final model?

Partial PROC GLMSELECT Output

Forward Selection Summary					
Step	Effect Entered	Number Effects In	Number Params In	F Value	Pr > F

0	Intercept	1	1	0.00	1.0000
<hr/>					
1	Type	2	6	77.64	<.0001
2	DriveTrain	3	8	44.53	<.0001
3	Origin	4	10	6.21	0.0022
4	Type*DriveTrain	5	16	3.32	0.0033

6. Fitting a Stepwise Regression Model

- a. Create a correlation matrix of the variables **MPG_Highway**, **HorsePower**, **EngineSize**, **Weight**, **WheelBase**, and **Length**. Which of the variables are positively correlated with **MPG_Highway**?

Partial PROC CORR Output

	MPG_Highway	Horsepower	Engine Size	Weight	Wheelbase	Length
MPG_Highway	1.00000	-0.64720	-0.71730	-0.79099	-0.52466	-0.46609
MPG (Highway)		<.0001	<.0001	<.0001	<.0001	<.0001
Horsepower	-0.64720	1.00000	0.78743	0.63080	0.38740	0.38155
	<.0001		<.0001	<.0001	<.0001	<.0001
EngineSize	-0.71730	0.78743	1.00000	0.80787	0.63652	0.63745
Engine Size (L)	<.0001		<.0001	<.0001	<.0001	<.0001
Weight	-0.79099	0.63080	0.80787	1.00000	0.76070	0.69002
Weight (LBS)	<.0001		<.0001	<.0001	<.0001	<.0001
Wheelbase	-0.52466	0.38740	0.63652	0.76070	1.00000	0.88919
Wheelbase (IN)	<.0001		<.0001	<.0001		<.0001
Length	-0.46609	0.38155	0.63745	0.69002	0.88919	1.00000
Length (IN)	<.0001		<.0001	<.0001	<.0001	

- b. Generate a stepwise selection model using **MPG_Highway** as the dependent variable. In addition, use the AICC selection criteria for generating the model. Save the final model with a STORE statement. Which variables are included in the final model?

Partial PROC GLMSELECT Output

Parameter Estimates				
Parameter	DF	Estimate	Standard Error	t Value
Intercept	1	39.179650	2.513074	15.59
Horsepower	1	-0.014254	0.003741	-3.81
EngineSize	1	-0.609518	0.323393	-1.88
Weight	1	-0.005206	0.000388	-13.41
Length	1	0.060715	0.016224	3.74

- c. The data set **Newdata_Cars** contains information about 10 different automobiles. Use PROC PLM to score the new data set. Request the predicted values, confidence limits, and prediction limits.

- d. Print the predicted values, the intervals, and the response and the model effects.

Obs	MPG_Highway		Engine Size		Weight	Length	Predicted	UCLM	LCLM	LCL	UCL
	Highway	Horsepower									
1	34	103	1.6	2348	153	33.8029	34.6824	32.9233	27.2641	40.3417	
2	26	194	3.5	3651	192	26.9326	27.3608	26.5044	20.4391	33.4261	
3	22	195	3.5	4802	194	21.0482	21.9232	20.1732	14.5100	27.5864	
4	33	104	1.6	2447	167	34.1233	34.7659	33.4806	27.6121	40.6344	
5	25	239	4.6	4474	221	23.0972	23.9596	22.2349	16.5607	29.6338	
6	26	185	3.4	3948	201	26.1222	26.6537	25.5908	19.6211	32.6234	
7	29	228	2.7	2811	170	29.9726	30.5497	29.3954	23.4675	36.4776	
8	26	143	2.2	3381	181	29.1896	29.7053	28.6739	22.6898	35.6895	
9	27	155	2.5	3380	188	29.2659	29.7158	28.8161	22.7709	35.7609	
10	33	157	2.4	3175	193	30.6691	31.2223	30.1158	24.1661	37.1720	

7. Fitting a Polynomial Model

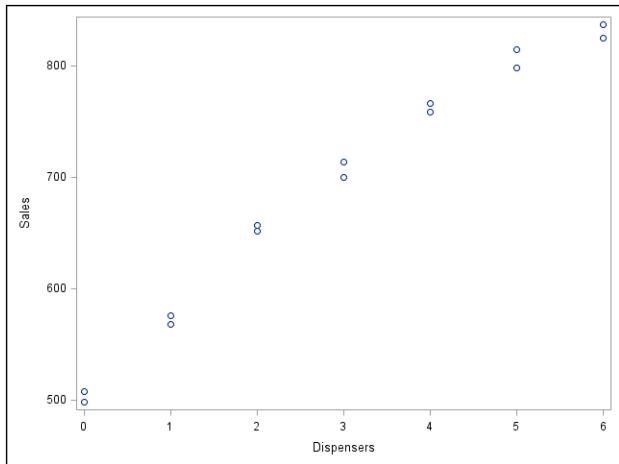
The **Cafeteria** data set consists of 14 cafeterias that are similar in terms of volume of business, type of clientele, and location. The number of self-service dispensers was assigned randomly at each cafeteria and the sales were recorded. The goal is to identify whether the sales of coffee are related to the number of self-service dispensers in a cafeteria line. The data set consists of the following data variables:

- Cafeteria** Cafeteria identification number
Dispensers Number of dispensers at each cafeteria
Sales Coffee sales in hundreds of gallons

- a. Print the **Cafeteria** data set. What is the range of the number of dispensers for a single cafeteria?

Obs	Cafeteria	Dispensers	Sales
1	1	0	507.9
2	2	0	498.0
3	3	1	568.3
4	4	1	575.5
5	5	2	651.6
6	6	2	657.1
7	7	3	713.8
8	8	3	699.6
9	9	4	758.4
10	10	4	765.7
11	11	5	797.7
12	12	5	814.3
13	13	6	836.8
14	14	6	825.1

- b. Create a scatter plot with **Sales** on the Y axis and **Dispensers** on the X axis. Do the **Sales** appear to be nonlinearly associated with the number of **Dispensers**?

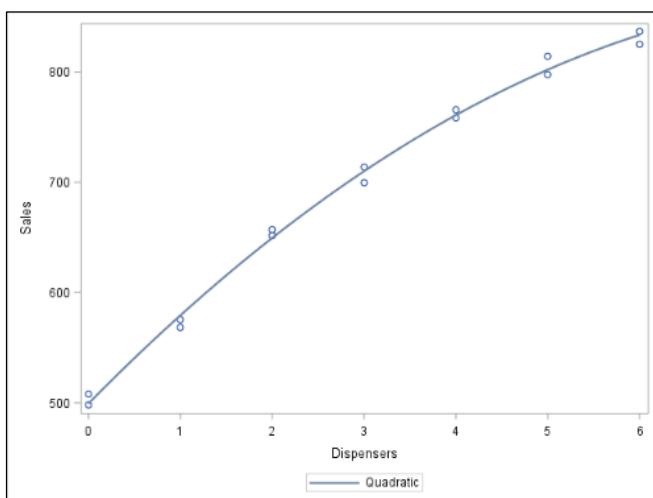


- c. Use PROC GLMSELECT to create a polynomial regression model. Be sure to use the OUTDESIGN= and PLOT=ALL options in the PROC GLMSELECT statement. Use the EFFECT statement to create a cubic polynomial model with **Dispensers** as the independent variable. Conduct forward selection and use a significance level of 0.05 as the barrier to entry. In addition, use the HIERARCHY=SINGLE option in the MODEL statement to enforce hierarchy. This ensures that higher order terms are in the model only if their corresponding lower order terms are in the model. Which order polynomial does the model select?

Partial PROC GLMSELECT Output

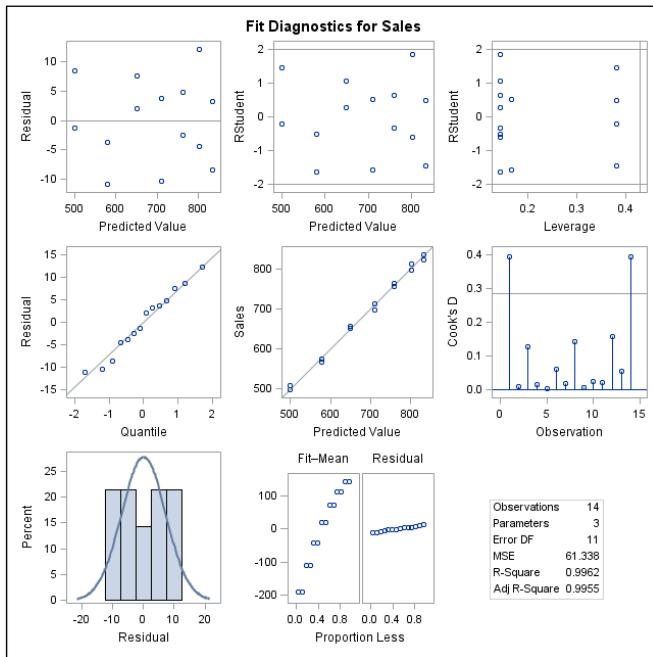
Parameter Estimates				
Parameter	DF	Estimate	Standard Error	t Value
Intercept	1	499.371429	4.833914	103.31
Dispensers	1	84.746429	3.773473	22.46
Dispensers ^{^2}	1	-4.839286	0.604239	-8.01

- d. Use PROC SGPLOT to create a scatter plot and add a line to the degree specified by the final selection model.



- e. Use the macro variable (**&_GLSMOD**) and the specified OUTDESIGN= data table to create PROC REG output. Notice that the **GLSMOD** macro variable stores only the significant variables.

Partial PROC REG Output



8. Fitting a Logistic Regression Model

The **Safety** data set is created by an insurance company that wants to relate the safety of vehicles to various features. The data set consists of the following variables:

Unsafe 1=Yes, 0=No

Size Size of vehicle (1, 2, or 3)

Weight Weight of Vehicle (1, 2, 3, 4, 5, or 6)

Region Asia or North America

Type Vehicle type (Large, Medium, Small, Sport/Utility, or Sports)

- a. Use PROC FREQ to create a table of each variable. What percentage of vehicles in the sample are rated as safe?

The FREQ Procedure					
Unsafe	Frequency	Percent	Cumulative		Cumulative Percent
			Frequency	Percent	
0	66	68.75	66	68.75	
1	30	31.25	96	100.00	

Size	Frequency	Percent	Cumulative		Cumulative Percent
			Frequency	Percent	
1	35	36.46	35	36.46	
2	29	30.21	64	66.67	
3	32	33.33	96	100.00	

Weight	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	1	1.04	1	1.04
2	11	11.46	12	12.50
3	53	55.21	65	67.71
4	26	27.08	91	94.79
5	3	3.13	94	97.92
6	2	2.08	96	100.00
Region	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Asia	35	36.46	35	36.46
N America	61	63.54	96	100.00
Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Large	16	16.67	16	16.67
Medium	29	30.21	45	46.88
Small	20	20.83	65	67.71
Sport/Utility	16	16.67	81	84.38
Sports	15	15.63	96	100.00

- b. Use PROC LOGISTIC to model the unsafety of a vehicle with independent variables **Weight**, **Region**, and **Size**. Specify **Region** and **Size** as categorical variables with reference categories *Asia* and *3* respectively. Request only the effect plot for the analysis and use the CLODDS=WALD option in the MODEL statement. Use the ESTIMATE statement to estimate the probability of a vehicle from North America with **Weight**=4 and **Size**=1 as unsafe.

Partial PROC LOGISTIC Output

Analysis of Maximum Likelihood Estimates					
Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	0.0500	1.8008	0.0008	0.9778
Weight	1	-0.6678	0.4589	2.1176	0.1456
Region N America	1	-0.3775	0.5624	0.4506	0.5020
Size 1	1	2.6783	0.8810	9.2422	0.0024
Size 2	1	0.6582	0.9231	0.5085	0.4758

- c. Because the significance of the parameter estimates were weak for the model above, conduct backward selection with a significance level to stay in the model of 0.05. Use ODS SELECT to request only the ModelBuildingSummary, ModelAnova, and ParameterEstimates values of the final model. Which variables were removed from the model?

Summary of Backward Elimination					
Step	Effect Removed	DF	Number In	Wald Chi-Square	Pr > ChiSq
1	Region	1	2	0.4506	0.5020
2	Weight	1	1	2.1565	0.1420

Type 3 Analysis of Effects					
Effect	DF	Chi-Square	Pr > ChiSq	Wald	
Size	2	24.2875	<.0001		
Analysis of Maximum Likelihood Estimates					
Parameter	DF	Estimate	Standard Error	Chi-Square	Pr > ChiSq
Intercept	1	-2.7080	0.7303	13.7505	0.0002
Size	1	3.3585	0.8125	17.0880	<.0001
Size	2	1.1393	0.8803	1.6751	0.1956

9. Fitting a Generalized Linear Model

A survey was undertaken to examine which factors are related to ear infections among swimmers. The response variable is the number of self-diagnosed ear infections reported by the participant. The data are stored in the **EarInfection** data set. The data set consists of the following variables:

Infections	Number of self-diagnosed ear infections
Swimmer	Frequent or occasional swimmer
Location	Typical swimming location (NonBeach or Beach)
Age	Age in years
Gender	Gender of swimmer (Male or Female)

 The data were obtained with permission from the OZDATA website. This website is a collection of data sets and is maintained in Australia.

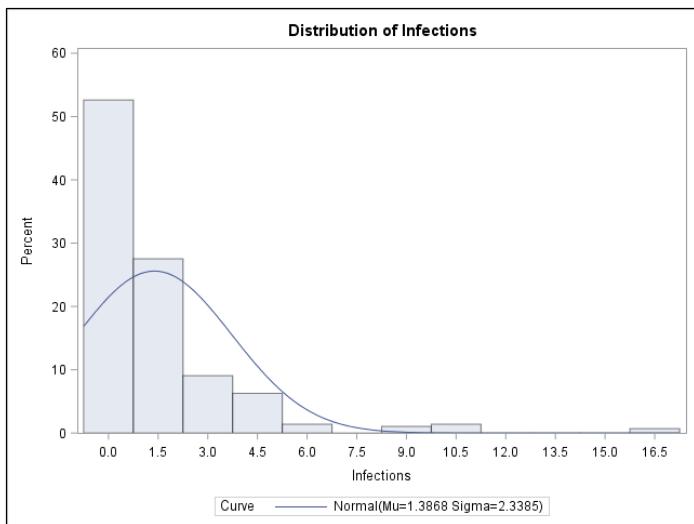
- a. Create a frequency table for the variables **Swimmer**, **Location**, **Age**, **Gender**, and **Infections**. What is the range of the number of infections for swimmers in this sample?

Swimmer	Frequency	Percent	Cumulative	Cumulative
			Frequency	Percent
Freq	143	49.83	143	49.83
Occas	144	50.17	287	100.00
Location	Frequency	Percent	Cumulative	Cumulative
			Frequency	Percent
Beach	147	51.22	147	51.22
NonBeach	140	48.78	287	100.00
Age	Frequency	Percent	Cumulative	Cumulative
			Frequency	Percent
15	28	9.76	28	9.76
16	26	9.06	54	18.82
17	28	9.76	82	28.57
18	32	11.15	114	39.72
19	26	9.06	140	48.78
20	16	5.57	156	54.36
21	15	5.23	171	59.58

22	16	5.57	187	65.16
23	16	5.57	203	70.73
24	16	5.57	219	76.31
25	12	4.18	231	80.49
26	14	4.88	245	85.37
27	14	4.88	259	90.24
28	15	5.23	274	95.47
29	13	4.53	287	100.00
Gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Female	99	34.49	99	34.49
Male	188	65.51	287	100.00
Infections	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	151	52.61	151	52.61
1	40	13.94	191	66.55
2	39	13.59	230	80.14
3	26	9.06	256	89.20
4	13	4.53	269	93.73
5	5	1.74	274	95.47
6	4	1.39	278	96.86
9	3	1.05	281	97.91
10	3	1.05	284	98.95
11	1	0.35	285	99.30
16	1	0.35	286	99.65
17	1	0.35	287	100.00

- b. Request only the BasicMeasures table and a histogram with a normal density estimate from PROC UNIVARIATE for the **Infections** variable. Does this variable appear to be normally distributed?

Basic Statistical Measures			
Location		Variability	
Mean	1.386760	Std Deviation	2.33854
Median	0.000000	Variance	5.46878
Mode	0.000000	Range	17.00000
		Interquartile Range	2.00000



- c. Create a Poisson regression model with **Infections** as the dependent variable and **Swimmer**, **Location**, **Age**, and **Gender** as the independent variables. For the variables **Swimmer**, **Location**, and **Gender** use the reference categories Occas, NonBeach, and Female respectively. Be sure to use the STORE statement to predict the number of Infections using PROC PLM. Which variables are statistically significant?

Partial PROC GENMOD Output

Analysis Of Maximum Likelihood Parameter Estimates							
Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits	Chi-Square	Pr > ChiSq	
Intercept	1	1.3586	0.2736	0.8224 1.8948	24.66	<.0001	
Swimmer Freq	1	-0.6086	0.1050	-0.8145 -0.4028	33.59	<.0001	
Location Beach	1	-0.4896	0.1048	-0.6951 -0.2841	21.81	<.0001	
Age	1	-0.0261	0.0122	-0.0500 -0.0021	4.55	0.0330	
Gender Male	1	-0.0294	0.1092	-0.2433 0.1846	0.07	0.7878	
Scale	0	1.0000	0.0000	1.0000 1.0000			

- d. Create a new data set with the following observations:

Swimmer	Location	Age	Gender
Freq	NonBeach	25	Female
Occas	Beach	15	Male

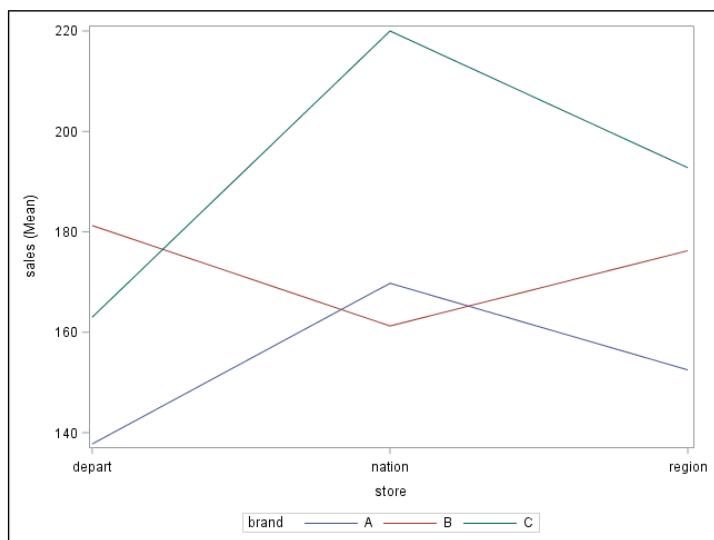
- e. Use PROC PLM to score the new data set and predict the number of **Infections** on the original data scale. Finally, print the predicted values from the PROC PLM output data set.

Obs	Swimmer	Location	Age	Gender	Predicted
1	Freq	NonBeach	25	Female	1.10338
2	Occas	Beach	15	Male	1.56621

10. Fitting a Two-Way Mixed Model

A market researcher is interested in comparing the average monthly sales of three brands (**A**, **B**, and **C**) of washing machines at various types of stores. She wants to see whether certain brands of washing machines sell better at certain types of stores. There are three stores that she can investigate, although she wants to make inferences about the population of various types of stores. The types of stores are **depart**, **nation**, and **region**. The response is **sales**. The nine treatment combinations are replicated four times for a total of 36 observations. The data are stored in a SAS data set **washer**. The variables included in the data set are **brand**, **store**, and **sales**.

- a. Use PROC SGPlot to plot the average value of **sales** (y) versus **store** (x) for each **brand** (group). What do you conclude from the graph?



- b. Is **brand** a fixed effect or a random effect? How would you classify **store** and **brand*store**?
 c. Analyze this design using the MIXED procedure and use the REML method. What are the estimates of variance components? Is **brand** a significant factor?

Partial PROC MIXED Output

Cov	Parm	Estimate		
	store	3.4931		
	brand*store	339.57		
	Residual	197.91		
Fit Statistics				
	-2 Res Log Likelihood	288.0		
	AIC (Smaller is Better)	294.0		
	AICC (Smaller is Better)	294.9		
	BIC (Smaller is Better)	291.3		
Type 3 Tests of Fixed Effects				
Effect	Num DF	Den DF	F Value	Pr > F
brand	2	4	2.87	0.1687

- d. Add an ESTIMATE statement to compare the average monthly sales of **brand A** and **brand B**.

Partial PROC MIXED Output

Label	Estimate	Standard		DF	t Value	Pr > t
		Error				
brand A vs brand B	-19.5833	16.1048		4	-1.22	0.2908

- e. Use the Expected Mean Squares table to test the variance components.

Partial PROC MIXED Output

Type 3 Analysis of Variance				
Source	DF	Sum of Squares	Mean Square	Expected Mean Square
brand	2	8932.722222	4466.361111	Var(Residual) + 4 Var(brand*store) + Q(brand)
store	2	3196.222222	1598.111111	Var(Residual) + 4 Var(brand*store) + 12 Var(store)
brand*store	4	6224.777778	1556.194444	Var(Residual) + 4 Var(brand*store)
Residual	27	5343.500000	197.907407	Var(Residual)

Type 3 Analysis of Variance				
Source	Error Term	DF	F Value	Pr > F
brand	MS(brand*store)	4	2.87	0.1687
store	MS(brand*store)	4	1.03	0.4366
brand*store	MS(Residual)	27	7.86	0.0002
Residual

End of Exercises

6.5 Solutions

Solutions to Exercises

1. Fitting a Regression Model

Percentage of body fat, age, weight, height, and 10 body circumference measurements were recorded for 252 men by Dr. Roger W. Johnson of Calvin College in Minnesota. The data are in the **BodyFat** data set, which consists of the following variables:

Case	Case number
PctBodyFat2	Percent body fat using Siri's equation (495/density - 450)
Age	Age in years
Weight	Weight in pounds
Height	Height in inches
Neck	Neck circumference (cm)
Chest	Chest circumference (cm)
Abdomen	Abdomen circumference (cm)
Hip	Hip circumference (cm)
Thigh	Thigh circumference (cm)
Knee	Knee circumference (cm)
Ankle	Ankle circumference (cm)
Biceps	Extended biceps circumference (cm)
Forearm	Forearm circumference (cm)
Wrist	Wrist circumference (cm)

- a. Generate PROC CORR output for the variables **Height**, **Neck**, **Chest**, and **Weight**. Which variables are highly correlated with **Weight**?

```
proc corr data=sp4r.bodyfat;
  var height neck chest weight;
run;
```

The CORR Procedure						
4 Variables: Height Neck Chest Weight						
Simple Statistics						
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
Height	252	70.30754	2.60958	17718	64.00000	77.75000
Neck	252	37.99206	2.43091	9574	31.10000	51.20000
Chest	252	100.82421	8.43048	25408	79.30000	136.20000
Weight	252	178.92440	29.38916	45089	118.50000	363.15000

Pearson Correlation Coefficients, N = 252

Prob > r under H0: Rho=0				
	Height	Neck	Chest	Weight
Height	1.00000	0.32114 <.0001	0.22683 0.0003	0.48689 <.0001
Neck	0.32114 <.0001	1.00000	0.78484 <.0001	0.83072 <.0001
Chest	0.22683 0.0003	0.78484 <.0001	1.00000	0.89419 <.0001
Weight	0.48689 <.0001	0.83072 <.0001	0.89419 <.0001	1.00000

- b. Generate PROC UNIVARIATE output with the same variables. Use the ODS SELECT statement to request only the BasicMeasures table. Do any of the variables appear to be skewed judging only from the mean and median summary statistics?

```
ods select basicmeasures;
proc univariate data=sp4r.bodyfat;
  var height neck chest weight;
run;
```

The UNIVARIATE Procedure			
Variable: Height			
Basic Statistical Measures			
Location		Variability	
Mean	70.30754	Std Deviation	2.60958
Median	70.00000	Variance	6.80992
Mode	71.50000	Range	13.75000
		Interquartile Range	4.00000
The UNIVARIATE Procedure			
Variable: Neck			
Basic Statistical Measures			
Location		Variability	
Mean	37.99206	Std Deviation	2.43091
Median	38.00000	Variance	5.90934
Mode	38.50000	Range	20.10000
		Interquartile Range	3.05000
The UNIVARIATE Procedure			
Variable: Chest			
Basic Statistical Measures			
Location		Variability	
Mean	100.8242	Std Deviation	8.43048
Median	99.6500	Variance	71.07292

Mode	99.1000	Range	56.90000
		Interquartile Range	11.15000

Note: The mode displayed is the smallest of 2 modes with a count of 5.

The UNIVARIATE Procedure
Variable: Weight

Basic Statistical Measures

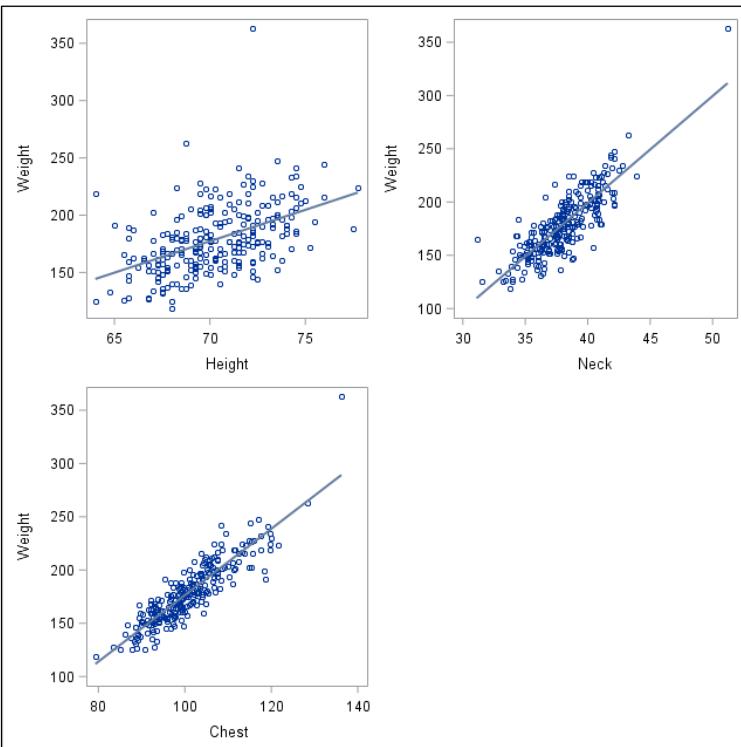
Location Variability

Mean	178.9244	Std Deviation	29.38916
Median	176.5000	Variance	863.72272
Mode	152.2500	Range	244.65000
		Interquartile Range	38.25000

Note: The mode displayed is the smallest of 9 modes with a count of 3.

- c. Use PROC SGSCATTER to plot **Weight** by **Height**, **Neck**, and **Chest** separately. Add the regression line to each plot. Does each variable appear to be linearly associated with **Weight**?

```
proc sgscatter data=sp4r.bodyfat;
  plot weight * (height neck chest) / reg;
run;
```



- d. Use PROC REG to create a multiple linear regression model with **Weight** as the dependent variable and **Height**, **Neck**, and **Chest** as independent variables. Use the ODS SELECT statement to request only the tables ANOVA, FitStatistics, and ParameterEstimates. Use the OUTPUT statement to create a new data set with the predicted and residual values. Which variables are statistically significant for this model?

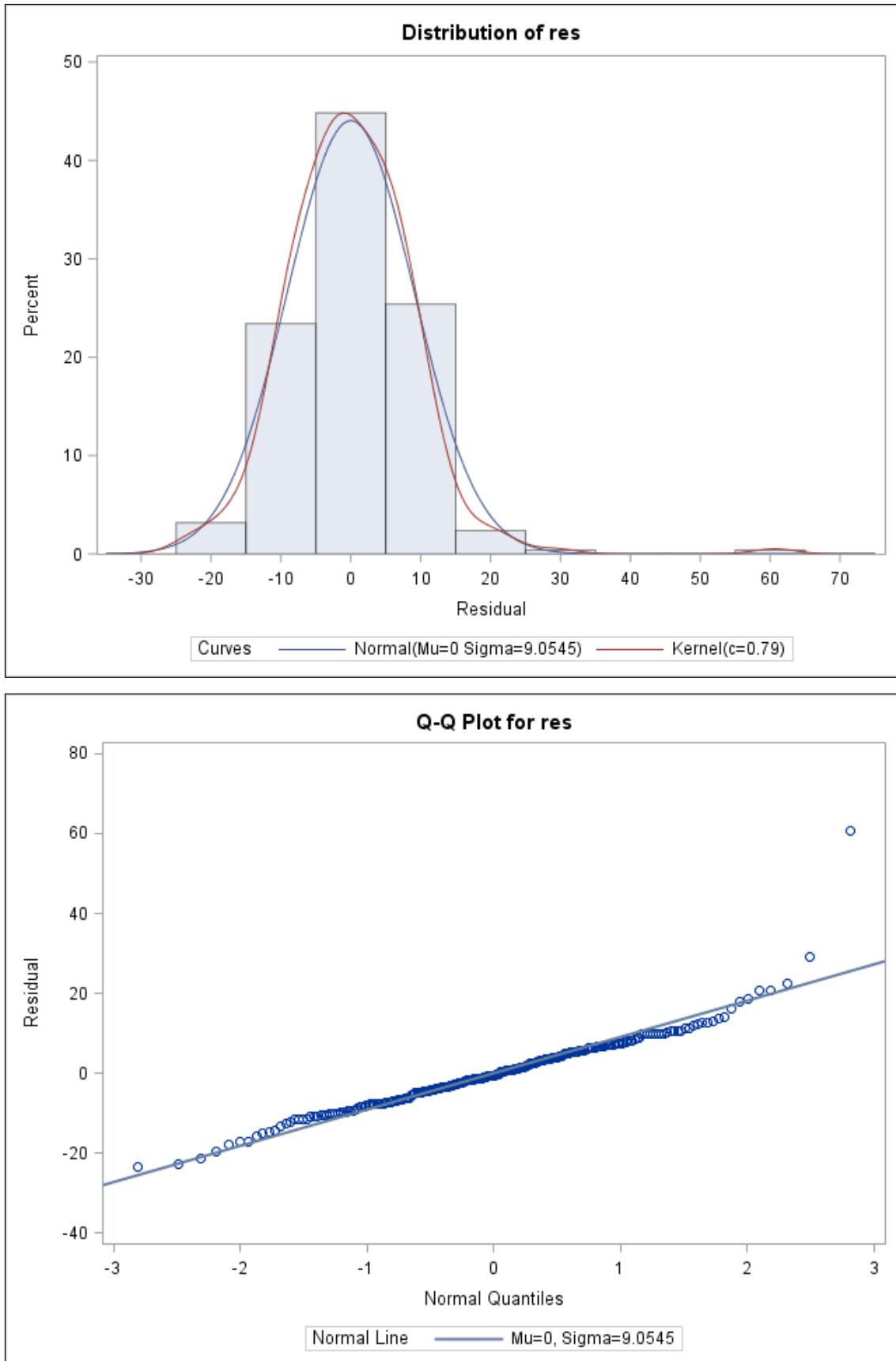
```
ods select anova fitstatistics parameterestimates;
```

```
proc reg data=sp4r.bodyfat;
  model weight = height neck chest;
  output out=sp4r.out predicted=pred residual=res;
run;quit;
```

The REG Procedure					
Model: MODEL1					
Dependent Variable: Weight					
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	3	196216	65405	788.25	<.0001
Error	248	20578	82.97585		
Corrected Total	251	216794			
Root MSE		9.10911	R-Square	0.9051	
Dependent Mean		178.92440	Adj R-Sq	0.9039	
Coeff Var		5.09104			
Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	1	-366.66231	16.07211	-22.81	<.0001
Height	1	2.96912	0.23287	12.75	<.0001
Neck	1	2.87083	0.39287	7.31	<.0001
Chest	1	2.25905	0.11015	20.51	<.0001

- e. Use PROC UNIVARIATE to create a histogram with a normal density estimate and a Q-Q plot compared to a normal distribution of the residuals. Use the ODS SELECT statement to request only the histogram and Q-Q plot. Do the residuals appear to be normally distributed?

```
ods select histogram qqplot;
proc univariate data=sp4r.out;
  var res;
  histogram res / normal kernel;
  qqplot res / normal(mu=est sigma=est);
run;
```



2. Predicting New Data

- a. Rerun the model from Exercise 1, but this time use a STORE statement to save the model.

```
proc reg data=sp4r.bodyfat;
  model weight = height neck chest;
  store mymod;
run;quit;
```

- b. The data set **Newdata_Bodyfat_Reg** contains five new observations. Use PROC PLM to score the new data. Use the PREDICTED keyword in the SCORE statement and save the scored data set as **Pred_Newdata_Bodyfat**.

```
proc plm restore=mymod;
  score data=sp4r.newdata_bodyfat_reg
    out=sp4r.pred_newdata_bodyfat predicted;
run;
```

The PLM Procedure

Store Information

Item Store	WORK.MYMOD
Data Set Created From	WORK.BODYFAT
Created By	PROC REG
Date Created	29NOV15:11:37:52
Response Variable	Weight
Model Effects	Intercept Height Neck Chest

- c. Print the predicted values from the scored data set and the response and independent variables.

```
proc print data=sp4r.pred_newdata_bodyfat;
  var weight height neck chest predicted;
run;
```

Obs	Weight	Height	Neck	Chest	Predicted
1	179.00	68.00	39.1	103.3	180.847
2	200.50	69.75	41.3	111.4	210.657
3	140.25	68.25	33.9	86.0	127.579
4	148.75	70.00	35.5	86.7	138.950
5	151.25	67.75	34.5	90.2	137.305

3. Fitting an ANOVA Model

The data set **Cars** contains information about a sample of 1993 model cars from the *1993 Cars Annual Auto Issue* published by *Consumer Reports* and from *Pace New Car and Truck 1993 Buying Guide*. The data set consists of the following variables:

Make	Name of the manufacturer
Model	Name of the model
Type	Vehicle type (Hybrid, SUV, Sedan, Sports, Truck, or Wagon)
Origin	Vehicle origin (Asia, Europe, or USA)
DriveTrain	Drivetrain type (All, Front, or Rear)
Invoice	Invoice price

MSRP	Manufacturer's suggested retail price
EngineSize	Engine displacement size in liters
Cylinders	Number of Cylinders
Horsepower	Maximum horsepower
MPG_City	Average city miles per gallon (EPA rating)
MPG_Highway	Average highway miles per gallon (EPA rating)
Weight	Weight of vehicle in pounds
Wheelbase	Wheelbase in inches
Length	Length of the vehicle in inches

- a. Generate a frequency table for the variable **Type**. Are the counts of each vehicle in this sample evenly distributed?

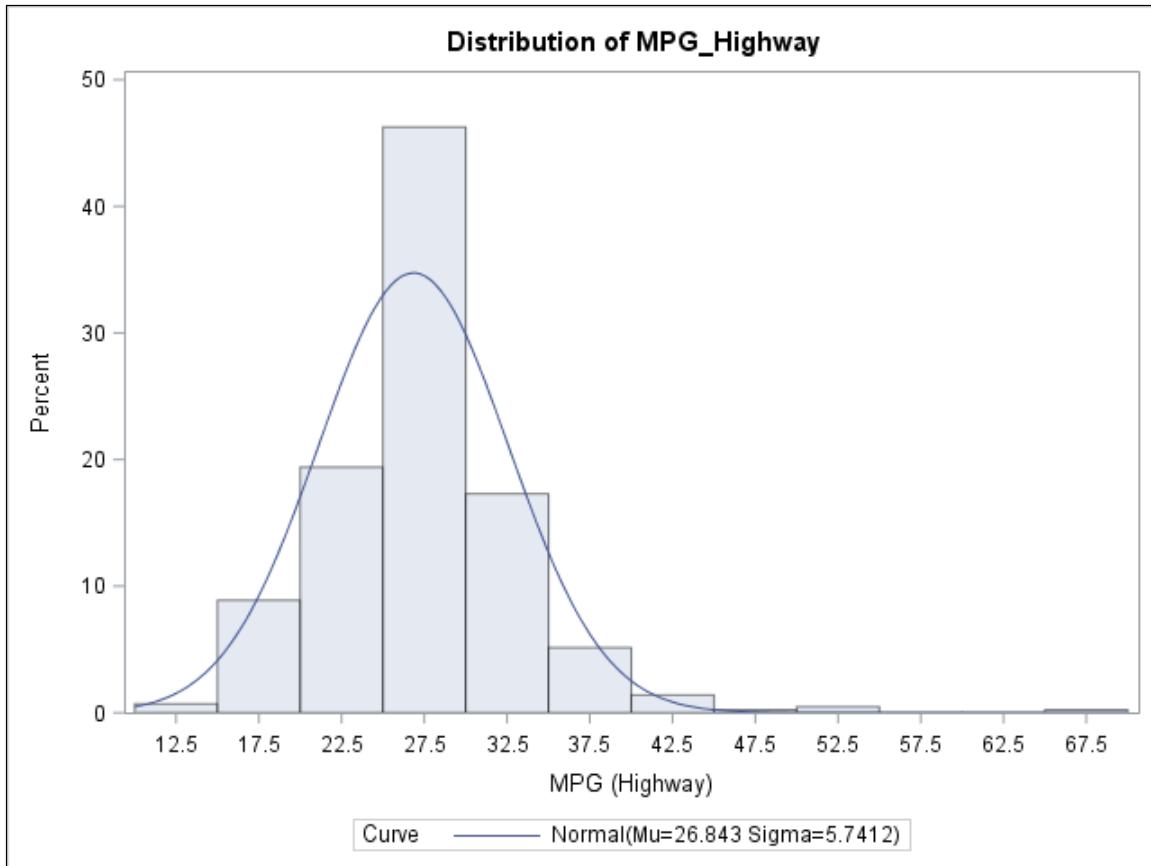
```
proc freq data=sp4r.cars;
  table type;
run;
```

The FREQ Procedure				
Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	0.70	3	0.70
SUV	60	14.02	63	14.72
Sedan	262	61.21	325	75.93
Sports	49	11.45	374	87.38
Truck	24	5.61	398	92.99
Wagon	30	7.01	428	100.00

- b. Use PROC UNIVARIATE to analyze the **MPG_Highway** variable. Request only the Moments table and the histogram plot with a density estimate. Does **MPG_Highway** appear to be normally distributed?

```
ods select moments histogram;
proc univariate data=sp4r.cars;
  var mpg_highway;
  histogram mpg_highway / normal;
run;
```

The UNIVARIATE Procedure			
Variable: MPG_Highway (MPG (Highway))			
Moments			
N	428	Sum Weights	428
Mean	26.8434579	Sum Observations	11489
Std Deviation	5.74120072	Variance	32.9613857
Skewness	1.25239527	Kurtosis	6.04561068
Uncorrected SS	322479	Corrected SS	14074.5117
Coeff Variation	21.3877092	Std Error Mean	0.27751141



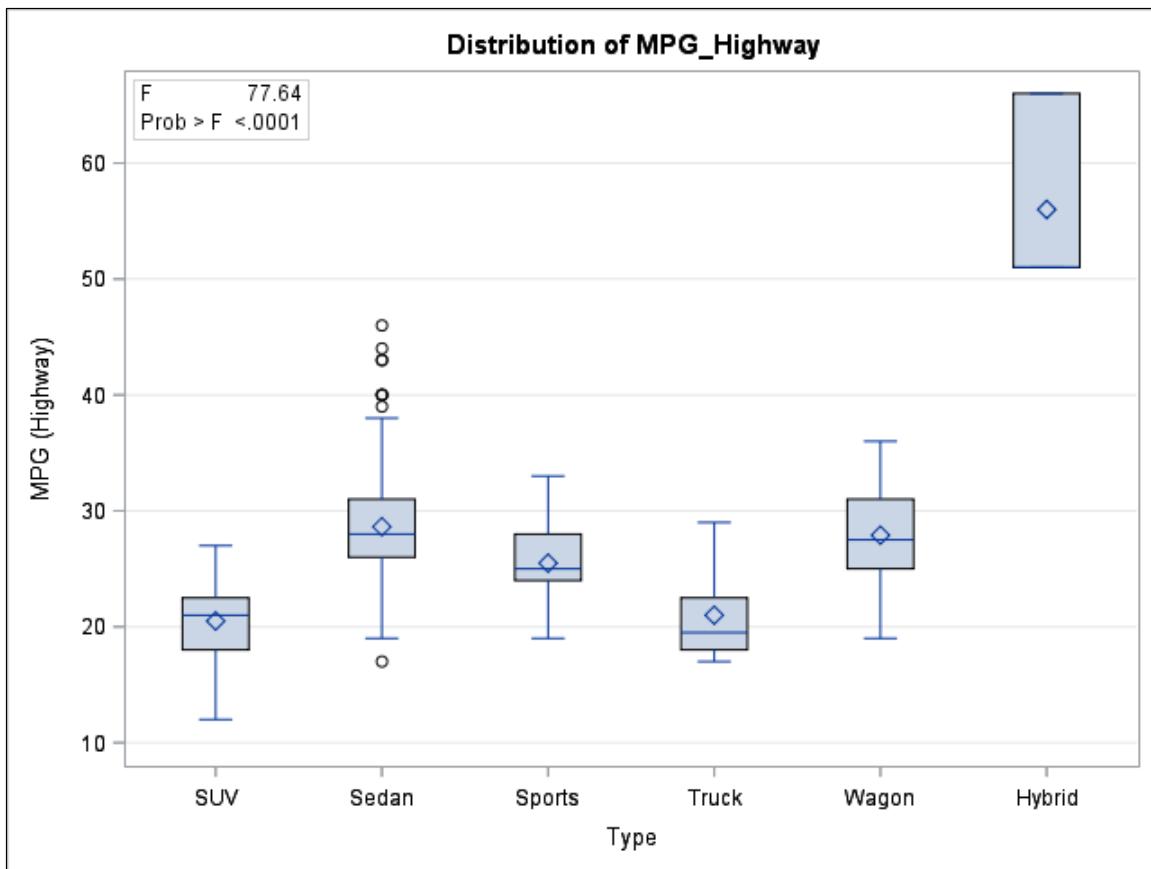
- c. Use PROC GLM to create a one-way ANOVA model. Use **MPG_Highway** as the dependent variable and **Type** as the independent variable. Use the PLOTS(ONLY)= option in the PROC GLM statement to request only the ANCOVA plot and use the hybrid vehicle as the reference category. Use the SOLUTION and CLPARM options in the MODEL statement to view parameter estimates and confidence limits. Identify the least squares means (LS-means). Use the ADJUST=TUKEY option and the PDIFF and CL options in the LSMEANS statement. Finally, use the ESTIMATE statement to see whether there is a significant difference in **MPG_Highway** for **SUV** versus **Truck**. Are all parameter estimates statistically significant? Are **SUV** and **Truck** significantly different?

```
proc glm data=sp4r.cars plots(only)=boxplot;
  class type (ref='Hybrid');
  model mpg_highway = type / solution clparm;
  lsmeans type / adjust=tukey pdiff cl;
  estimate 'SUV vs Truck' type 1 0 0 -1 0 0;
run;quit;
```

The GLM Procedure			
Class Level Information			
Class	Levels	Values	
Type	6	SUV Sedan Sports Truck Wagon Hybrid	
		Number of Observations Read	428
		Number of Observations Used	428

Dependent Variable: MPG_Highway MPG (Highway)						
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F	
Model	5	6743.47900	1348.69580	77.64	<.0001	
Error	422	7331.03268	17.37212			
Corrected Total	427	14074.51168				
R-Square	Coeff Var	Root MSE	MPG_Highway	Mean		
0.479127	15.52701	4.167987		26.84346		
Source	DF	Type I SS	Mean Square	F Value	Pr > F	
Type	5	6743.478998	1348.695800	77.64	<.0001	
Source	DF	Type III SS	Mean Square	F Value	Pr > F	
Type	5	6743.478998	1348.695800	77.64	<.0001	
Parameter	Estimate	Standard Error	t Value	Pr > t	95% Confidence Limits	
Intercept	56.00000000 B	2.40638840	23.27	<.0001	51.26999968	60.73000032
Type SUV	-35.50000000 B	2.46581434	-14.40	<.0001	-40.34680804	-30.65319196
Type Sedan	-27.37022901 B	2.42012622	-11.31	<.0001	-32.12723240	-22.61322562
Type Sports	-30.51020408 B	2.47895907	-12.31	<.0001	-35.38284942	-25.63755875
Type Truck	-35.00000000 B	2.55236033	-13.71	<.0001	-40.01692295	-29.98307705
Type Wagon	-28.10000000 B	2.52384144	-11.13	<.0001	-33.06086618	-23.13913382
Type Hybrid	0.00000000 B

NOTE: The X'X matrix has been found to be singular, and a generalized inverse was used to solve the normal equations. Terms whose estimates are followed by the letter 'B' are not uniquely estimable.



Least Squares Means						
Adjustment for Multiple Comparisons: Tukey-Kramer						
Type	MPG_Highway		LSMEAN			Number
	LSMEAN	LSMEAN	LSMEAN	LSMEAN	LSMEAN	
SUV	20.500000					1
Sedan	28.6297710					2
Sports	25.4897959					3
Truck	21.0000000					4
Wagon	27.9000000					5
Hybrid	56.0000000					6

Least Squares Means for effect Type						
Pr > t for H0: LSMean(i)=LSMean(j)						
Dependent Variable: MPG_Highway						

i/j	1	2	3	4	5	6
1		<.0001	<.0001	0.9963	<.0001	<.0001
2	<.0001		<.0001	<.0001	0.9443	<.0001
3	<.0001	<.0001		0.0003	0.1280	<.0001
4	0.9963	<.0001	0.0003		<.0001	<.0001
5	<.0001	0.9443	0.1280	<.0001		<.0001
6	<.0001	<.0001	<.0001	<.0001	<.0001	

MPG_Highway			
Type	LSMEAN	95% Confidence Limits	
SUV	20.500000	18.500000	22.500000
Sedan	28.6297710	26.6297710	30.6297710
Sports	25.4897959	23.4897959	27.4897959
Truck	21.0000000	18.0000000	24.0000000
Wagon	27.9000000	25.9000000	30.9000000
Hybrid	56.0000000	53.0000000	59.0000000

SUV	20.500000	19.442340	21.557660
Sedan	28.629771	28.123630	29.135912
Sports	25.489796	24.319424	26.660167
Truck	21.000000	19.327692	22.672308
Wagon	27.900000	26.404243	29.395757
Hybrid	56.000000	51.270000	60.730000

Least Squares Means for Effect Type

		Difference Between Means	Simultaneous 95% Confidence Limits for LSMean(i)-LSMean(j)	
i	j			
1	2	-8.129771	-9.837539	-6.422003
1	3	-4.989796	-7.287355	-2.692237
1	4	-0.500000	-3.381944	2.381944
1	5	-7.400000	-10.068162	-4.731838
1	6	-35.500000	-42.559293	-28.440707

Adjustment for Multiple Comparisons: Tukey-Kramer

		Difference Between Means	Simultaneous 95% Confidence Limits for LSMean(i)-LSMean(j)	
i	j			
2	3	3.139975	1.282775	4.997175
2	4	7.629771	5.084970	10.174572
2	5	0.729771	-1.570120	3.029662
2	6	-27.370229	-34.298723	-20.441735
3	4	4.489796	1.516864	7.462728
3	5	-2.410204	-5.176394	0.355986
3	6	-30.510204	-37.607128	-23.413280
4	5	-6.900000	-10.167817	-3.632183
4	6	-35.000000	-42.307062	-27.692938
5	6	-28.100000	-35.325416	-20.874584

Dependent Variable: MPG_Highway MPG (Highway)

Parameter	Estimate	Standard			
		Error	t Value	Pr > t	95% Confidence Limits
SUV vs Truck	-0.50000000	1.00666449	-0.50	0.6197	-2.47870110 1.47870110

4. Fitting an ANCOVA Model

- a. Extend the ANOVA model above to an ANCOVA model by adding the **HorsePower** variable into the set of independent variables. First, create a macro variable of the mean value of the **HorsePower** variable.

```
proc sql;
  select mean(horsepower) into :hp_mean from sp4r.cars;
quit;
```

215.8855

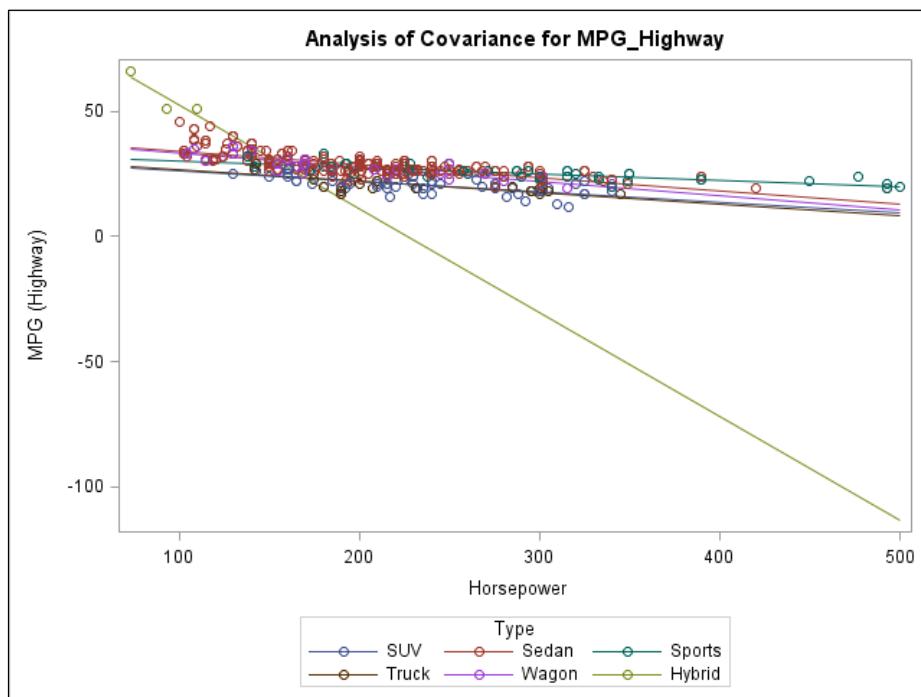
- b. Use PROC GLM to fit an ANCOVA model. Use the PLOTS(ONLY)= option to request only the ANCOVA plot from the GLM procedure. Identify the LS-means for the variable **Type**. Use the AT option to hold the variable **HorsePower** fixed at the mean and use the ADJUST=TUKEY option. Estimate the significance of the same linear combination as before, **SUV** versus **Truck**. Are all parameter estimates statistically significant? Is **SUV** still significantly different from **Truck**?

```
proc glm data=sp4r.cars plots(only)=ancovaplot;
  class type (ref='Hybrid');
  model mpg_highway = type|horsepower / solution clparm;
  lsmeans type / at horsepower=&hp_mean adjust=tukey pdiff cl;
  estimate 'SUV vs Truck' type 1 0 0 -1 0 0 type*horsepower
            &hp_mean 0 0 -&hp_mean 0 0;
run;quit;
```

The GLM Procedure					
Class Level Information					
Class	Levels	Values			
Type	6	SUV Sedan Sports Truck Wagon Hybrid			
Number of Observations Read			428		
Number of Observations Used			428		

Dependent Variable: MPG_Highway MPG (Highway)					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	10765.94182	978.72198	123.06	<.0001
Error	416	3308.56986	7.95329		
Corrected Total	427	14074.51168			
R-Square	Coeff Var	Root MSE	MPG_Highway Mean		
0.764925	10.50594	2.820158	26.84346		
Source	DF	Type I SS	Mean Square	F Value	Pr > F
Type	5	6743.478998	1348.695800	169.58	<.0001
Horsepower	1	3699.837286	3699.837286	465.20	<.0001
Horsepower*Type	5	322.625534	64.525107	8.11	<.0001
Source	DF	Type III SS	Mean Square	F Value	Pr > F
Type	5	653.3927903	130.6785581	16.43	<.0001
Horsepower	1	272.6094049	272.6094049	34.28	<.0001
Horsepower*Type	5	322.6255340	64.5251068	8.11	<.0001
Parameter	Estimate	Standard Error	t Value	Pr > t	

Intercept		94.22157434	B	10.03894199	9.39	<.0001
Type	SUV	-64.05940099	B	10.16284659	-6.30	<.0001
Type	Sedan	-55.15403535	B	10.05608869	-5.48	<.0001
Type	Sports	-61.56306301	B	10.12406223	-6.08	<.0001
Type	Truck	-62.82986720	B	10.28016825	-6.11	<.0001
Parameter		95% Confidence Limits				
Intercept		74.48819768 113.95495100				
Type	SUV	-84.03633480		-44.08246718		
Type	Sedan	-74.92111698		-35.38695372		
Type	Sports	-81.46375905		-41.66236696		
Type	Truck	-83.03741819		-42.62231620		



Dependent Variable: MPG_Highway MPG (Highway)						
Parameter		Estimate	Standard Error	t Value	Pr > t	
Type	Wagon	-55.28132488	B	10.17751744	-5.43	<.0001
Type	Hybrid	0.00000000	B	.	.	.
Horsepower		-0.41545190	B	0.10767414	-3.86	0.0001
Horsepower*Type	SUV	0.37447865	B	0.10787191	3.47	0.0006
Horsepower*Type	Sedan	0.36369176	B	0.10771002	3.38	0.0008
Horsepower*Type	Sports	0.39022444	B	0.10776347	3.62	0.0003
Horsepower*Type	Truck	0.36923229	B	0.10809315	3.42	0.0007
Horsepower*Type	Wagon	0.35854339	B	0.10798664	3.32	0.0010
Horsepower*Type	Hybrid	0.00000000	B	.	.	.
Parameter		95% Confidence Limits				
Type	Wagon	-75.28709693 -35.27555284				
Type	Hybrid	.				
Horsepower		-0.62710512 -0.20379867				

Horsepower*Type SUV	0.16243668	0.58652063
Horsepower*Type Sedan	0.15196801	0.57541550
Horsepower*Type Sports	0.17839563	0.60205326
Horsepower*Type Truck	0.15675543	0.58170915
Horsepower*Type Wagon	0.14627590	0.57081088
Horsepower*Type Hybrid	.	.

The GLM Procedure						
Least Squares Means at Horsepower=215.8855						
Adjustment for Multiple Comparisons: Tukey-Kramer						
Type	MPG_Highway		LSMEAN			
Type	LSMEAN	Number	LSMEAN	Number	LSMEAN	Number
SUV	21.3166445	1	21.3166445	1	21.3166445	1
Sedan	27.8932754	2	27.8932754	2	27.8932754	2
Sports	27.2122700	3	27.2122700	3	27.2122700	3
Truck	21.4135653	4	21.4135653	4	21.4135653	4
Wagon	26.6545290	5	26.6545290	5	26.6545290	5
Hybrid	4.5315343	6	4.5315343	6	4.5315343	6
Least Squares Means for effect Type						
Pr > t for H0: LSMean(i)=LSMean(j)						
Dependent Variable: MPG_Highway						
i/j	1	2	3	4	5	6
1		<.0001	<.0001	1.0000	<.0001	0.8125
2	<.0001		0.7970	<.0001	0.2596	0.5072
3	<.0001	0.7970		<.0001	0.9750	0.5414
4	1.0000	<.0001	<.0001		<.0001	0.8091
5	<.0001	0.2596	0.9750	<.0001		0.5692
6	0.8125	0.5072	0.5414	0.8091	0.5692	
Type	MPG_Highway		95% Confidence Limits			
Type	LSMEAN	LSMEAN	95% Confidence Limits	95% Confidence Limits	95% Confidence Limits	95% Confidence Limits
SUV	21.316645	21.316645	20.556637	22.076652	20.556637	22.076652
Sedan	27.893275	27.893275	27.542081	28.244470	27.542081	28.244470
Sports	27.212270	27.212270	26.225452	28.199088	26.225452	28.199088
Truck	21.413565	21.413565	20.269704	22.557427	20.269704	22.557427
Wagon	26.654529	26.654529	25.582575	27.726483	25.582575	27.726483
Hybrid	4.531534	4.531534	-21.883843	30.946912	-21.883843	30.946912
Least Squares Means for Effect Type						
i	j	Difference Between Means	Simultaneous 95% Confidence Limits for LSMean(i)-LSMean(j)			
1	2	-6.576631	-7.796069	-5.357193	-7.796069	-5.357193
1	3	-5.895626	-7.709809	-4.081442	-7.709809	-4.081442
1	4	-0.096921	-2.097200	1.903358	-2.097200	1.903358
1	5	-5.337884	-7.251807	-3.423962	-7.251807	-3.423962
1	6	16.785110	-21.705315	55.275535	-21.705315	55.275535

Least Squares Means at Horsepower=215.8855 Adjustment for Multiple Comparisons: Tukey-Kramer				
Least Squares Means for Effect Type				
i	j	Difference Between Means	Simultaneous 95% Confidence Limits for LSMean(i)-LSMean(j)	
2	3	0.681005	-0.844623	2.206633
2	4	6.479710	4.736897	8.222523
2	5	1.238746	-0.404232	2.881725
2	6	23.361741	-15.116163	61.839645
3	4	5.798705	3.598335	7.999075
3	5	0.557741	-1.564430	2.679912
3	6	22.680736	-15.820606	61.182078
4	5	-5.240964	-7.524265	-2.957662
4	6	16.882031	-21.628529	55.392591
5	6	22.122995	-16.383176	60.629165

The GLM Procedure						
Dependent Variable: MPG_Highway MPG (Highway)						
Parameter	Estimate	Standard Error	t Value	Pr > t	95% Confidence Limits	
SUV vs Truck	-0.09692078	0.69865201	-0.14	0.8897	-1.47024909	1.27640754

5. Fitting a Stepwise ANOVA Model

- a. Use the **Cars** data set to practice interacting with the GLMSELECT procedure. Create frequency tables for the variables **Type**, **Origin**, and **DriveTrain**.

```
proc freq data=sp4r.cars;
  tables type origin drivetrain;
run;
```

The FREQ Procedure					
Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent	
Hybrid	3	0.70	3	0.70	
SUV	60	14.02	63	14.72	
Sedan	262	61.21	325	75.93	
Sports	49	11.45	374	87.38	
Truck	24	5.61	398	92.99	
Wagon	30	7.01	428	100.00	
Origin	Frequency	Percent	Cumulative Frequency	Cumulative Percent	
Asia	158	36.92	158	36.92	
Europe	123	28.74	281	65.65	
USA	147	34.35	428	100.00	
Drive			Cumulative	Cumulative	

Train	Frequency	Percent	Frequency	Percent
All	92	21.50	92	21.50
Front	226	52.80	318	74.30
Rear	110	25.70	428	100.00

- b. Create a model with **MPG_Highway** as the dependent variable and the independent variables **Type**, **Origin**, and **DriveTrain** with all possible interactions. Specify forward selection and let the significance level to entry be 0.05. In addition, use the HIERARCHY=SINGLE option in the MODEL statement to enforce hierarchy. Which parameters were chosen for the final model?

```
proc glmselect data=sp4r.cars outdesign=sp4r.des;
  class type origin drivetrain;
  model mpg_highway = type|origin|drivetrain / selection=forward
    select=sl sle=.05 hierarchy=single;
run;quit;
```

Data Set	WORK.CARS
Dependent Variable	MPG_Highway
Selection Method	Forward
Select Criterion	Significance Level
Stop Criterion	Significance Level
Entry Significance Level (SLE)	0.05
Effect Hierarchy Enforced	Single
Number of Observations Read	428
Number of Observations Used	428
Class Level Information	
Class	Levels Values
Type	6 Hybrid SUV Sedan Sports Truck Wagon
Origin	3 Asia Europe USA
DriveTrain	3 All Front Rear
Dimensions	
Number of Effects	8
Number of Parameters	86

The GLMSELECT Procedure						
Forward Selection Summary						
Step	Effect Entered	Number Effects In	Number Params In	F Value	Pr > F	
0	Intercept	1	1	0.00	1.0000	
1	Type	2	6	77.64	<.0001	
2	DriveTrain	3	8	44.53	<.0001	
3	Origin	4	10	6.21	0.0022	
4	Type*DriveTrain	5	16	3.32	0.0033	
Selection stopped as the candidate for entry has SLE > 0.05.						

Stop Details					
Candidate For	Effect	Candidate Significance	Compare Significance		
Entry	Origin*DriveTrain	0.2121	>	0.0500	(SLE)

The GLMSELECT Procedure Selected Model					
The selected model is the model at the last step (Step 4).					
Effects: Intercept Type Origin DriveTrain Type*DriveTrain					
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	
Model	15	8471.88736	564.79249	41.53	
Error	412	5602.62433	13.59860		
Corrected Total	427	14075			
Root MSE 3.68763					
Dependent Mean 26.84346					
R-Square 0.6019					
Adj R-Sq 0.5874					
AIC 1562.75916					
AICC 1564.25184					
SBC 1197.70513					

Parameter Estimates					
Parameter		DF	Estimate	Standard Error	t Value
Intercept		1	25.326780	1.422054	17.81
Type	Hybrid	1	24.480338	2.362045	10.36
Type	SUV	1	-9.537231	1.270761	-7.51
Type	Sedan	1	-0.256653	1.485156	-0.17
Type	Sports	1	-0.773228	1.526241	-0.51
Type	Truck	1	-3.018065	1.764211	-1.71
Type	Wagon	0	0	.	.
Origin	Asia	1	1.431807	0.432855	3.31
Origin	Europe	1	0.208559	0.497170	0.42
Origin	USA	0	0	.	.
DriveTrain	All	1	-0.856783	1.863473	-0.46
DriveTrain	Front	1	4.761075	1.707919	2.79
DriveTrain	Rear	0	0	.	.
Type*DriveTrain	Hybrid Front	0	0	.	.
Type*DriveTrain	SUV All	1	4.541063	1.860482	2.44
Type*DriveTrain	SUV Front	0	0	.	.
Type*DriveTrain	Sedan All	1	0.687488	2.047517	0.34
Type*DriveTrain	Sedan Front	1	-0.367167	1.811440	-0.20
Type*DriveTrain	Sedan Rear	0	0	.	.
Type*DriveTrain	Sports All	1	2.005373	2.560593	0.78

Type*DriveTrain Sports Front	1	-3.386646	2.242753	-1.51
Type*DriveTrain Sports Rear	0	0	.	.
Type*DriveTrain Truck All	1	-2.715185	2.394578	-1.13
Type*DriveTrain Truck Rear	0	0	.	.
Type*DriveTrain Wagon All	0	0	.	.
Type*DriveTrain Wagon Front	0	0	.	.
Type*DriveTrain Wagon Rear	0	0	.	.

6. Fitting a Stepwise Regression Model

- a. Create a correlation matrix of the variables **MPG_Highway**, **HorsePower**, **EngineSize**, **Weight**, **WheelBase**, and **Length**. Which of the variables are positively correlated with **MPG_Highway**?

```
proc corr data=sp4r.cars;
  var mpg_highway horsepower enginesize weight wheelbase length;
run;
```

The CORR Procedure							
6 Variables: MPG_Highway Horsepower EngineSize Weight Wheelbase Length							
Simple Statistics							
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum	Label
MPG_Highway	428	26.84346	5.74120	11489	12.00000	66.00000	MPG (Highway)
Horsepower	428	215.88551	71.83603	92399	73.00000	500.00000	
EngineSize	428	3.19673	1.10859	1368	1.30000	8.30000	Engine Size (L)
Weight	428	3578	758.98321	1531364	1850	7190	Weight (LBS)
Wheelbase	428	108.15421	8.31181	46290	89.00000	144.00000	Wheelbase (IN)
Length	428	186.36215	14.35799	79763	143.00000	238.00000	Length (IN)
Pearson Correlation Coefficients, N = 428							
		Prob > r under H0: Rho=0					
	MPG_Highway	Horsepower	EngineSize	Weight	Wheelbase	Length	
MPG_Highway	1.00000	-0.64720	-0.71730	-0.79099	-0.52466	-0.46609	
MPG (Highway)		<.0001	<.0001	<.0001	<.0001	<.0001	<.0001
Horsepower	-0.64720	1.00000	0.78743	0.63080	0.38740	0.38155	
	<.0001		<.0001	<.0001	<.0001	<.0001	<.0001
EngineSize	-0.71730	0.78743	1.00000	0.80787	0.63652	0.63745	
Engine Size (L)	<.0001	<.0001		<.0001	<.0001	<.0001	<.0001
Weight	-0.79099	0.63080	0.80787	1.00000	0.76070	0.69002	
Weight (LBS)	<.0001	<.0001	<.0001		<.0001	<.0001	<.0001
Wheelbase	-0.52466	0.38740	0.63652	0.76070	1.00000	0.88919	
Wheelbase (IN)	<.0001	<.0001	<.0001	<.0001		<.0001	<.0001
Length	-0.46609	0.38155	0.63745	0.69002	0.88919	1.00000	
Length (IN)	<.0001	<.0001	<.0001	<.0001	<.0001		<.0001

- b. Generate a stepwise selection model using **MPG_Highway** as the dependent variable. In addition, use the AICC selection criteria for generating the model. Save the final model with a STORE statement. Which variables are included in the final model?

```
proc glmselect data=sp4r.cars outdesign=sp4r.des;
  model mpg_highway = horsepower enginesize weight wheelbase length /
    selection=stepwise select=AICC;
  store mymod;
run;quit;
```

The GLMSELECT Procedure	
Data Set	WORK.CARS
Dependent Variable	MPG_Highway
Selection Method	Stepwise
Select Criterion	AICC
Stop Criterion	AICC
Effect Hierarchy Enforced	None
Number of Observations Read	428
Number of Observations Used	428
Dimensions	
Number of Effects	6
Number of Parameters	6

The GLMSELECT Procedure				
Stepwise Selection Summary				
Step	Effect Entered	Effect Removed	Number Effects In	AICC
0	Intercept		1	1927.0312
1	Weight		2	1508.5057
2	Horsepower		3	1466.6351
3	Length		4	1457.7566
4	EngineSize		5	1456.2347*

* Optimal Value of Criterion

Selection stopped at a local minimum of the AICC criterion.

Stop Details

Candidate For	Effect	Candidate AICC	Compare AICC
Entry	Wheelbase	1457.8076	> 1456.2347
Removal	EngineSize	1457.7566	> 1456.2347

Selected Model				
The selected model is the model at the last step (Step 4).				
Effects: Intercept Horsepower EngineSize Weight Length				
Analysis of Variance				
Source	DF	Sum of Squares	Mean Square	F Value
Model	4	9478.07276	2369.51819	218.06
Error	423	4596.43892	10.86629	
Corrected Total	427	14075		
Root MSE 3.29640				
Dependent Mean 26.84346				
R-Square 0.6734				
Adj R-Sq 0.6703				
AIC 1456.03517				
AICC 1456.23469				
SBC 1046.33078				

Selected Model				
Parameter Estimates				
Parameter	DF	Estimate	Standard Error	t Value
Intercept	1	39.179650	2.513074	15.59
Horsepower	1	-0.014254	0.003741	-3.81
EngineSize	1	-0.609518	0.323393	-1.88
Weight	1	-0.005206	0.000388	-13.41
Length	1	0.060715	0.016224	3.74

- c. The data set **Newdata_Cars** contains information about 10 different automobiles. Use PROC PLM to score the new data set. Request the predicted values, confidence limits, and prediction limits.

```
proc plm restore=mymod;
  score data=sp4r.newdata_cars out=sp4r.pred_newdata_cars
    predicted uclm lclm lcl ucl;
run;
```

The PLM Procedure	
Store Information	
Item Store	WORK.MYMOD
Data Set Created From	WORK.CARS
Created By	PROC GLMSELECT
Date Created	29NOV15:12:23:50
Response Variable	MPG_Highway
Model Effects	Intercept Horsepower EngineSize Weight Length



WheelBase is not included in the Model Effects because it was removed during the selection process.

- d. Print the predicted values, the intervals, and the response and the model effects.

```
proc print data=sp4r.pred_newdata_cars;
  var mpg_highway &_glsmmod predicted uclm lclm lcl ucl;
run;
```

Obs	MPG_Highway		Engine		Predicted	UCLM	LCLM	LCL	UCL	
	Highway	Horsepower	Size	Weight						
1	34	103	1.6	2348	153	33.8029	34.6824	32.9233	27.2641	40.3417
2	26	194	3.5	3651	192	26.9326	27.3608	26.5044	20.4391	33.4261
3	22	195	3.5	4802	194	21.0482	21.9232	20.1732	14.5100	27.5864
4	33	104	1.6	2447	167	34.1233	34.7659	33.4806	27.6121	40.6344
5	25	239	4.6	4474	221	23.0972	23.9596	22.2349	16.5607	29.6338
6	26	185	3.4	3948	201	26.1222	26.6537	25.5908	19.6211	32.6234
7	29	228	2.7	2811	170	29.9726	30.5497	29.3954	23.4675	36.4776
8	26	143	2.2	3381	181	29.1896	29.7053	28.6739	22.6898	35.6895
9	27	155	2.5	3380	188	29.2659	29.7158	28.8161	22.7709	35.7609
10	33	157	2.4	3175	193	30.6691	31.2223	30.1158	24.1661	37.1720

7. Fitting a Polynomial Model

The **Cafeteria** data set consists of 14 cafeterias that are similar in terms of volume of business, type of clientele, and location. The number of self-service dispensers was assigned randomly at each cafeteria and the sales were recorded. The goal is to identify whether the sales of coffee are related to the number of self-service dispensers in a cafeteria line. The data set consists of the following data variables:

Cafeteria Cafeteria identification number

Dispensers Number of dispensers at each cafeteria

Sales Coffee sales in hundreds of gallons

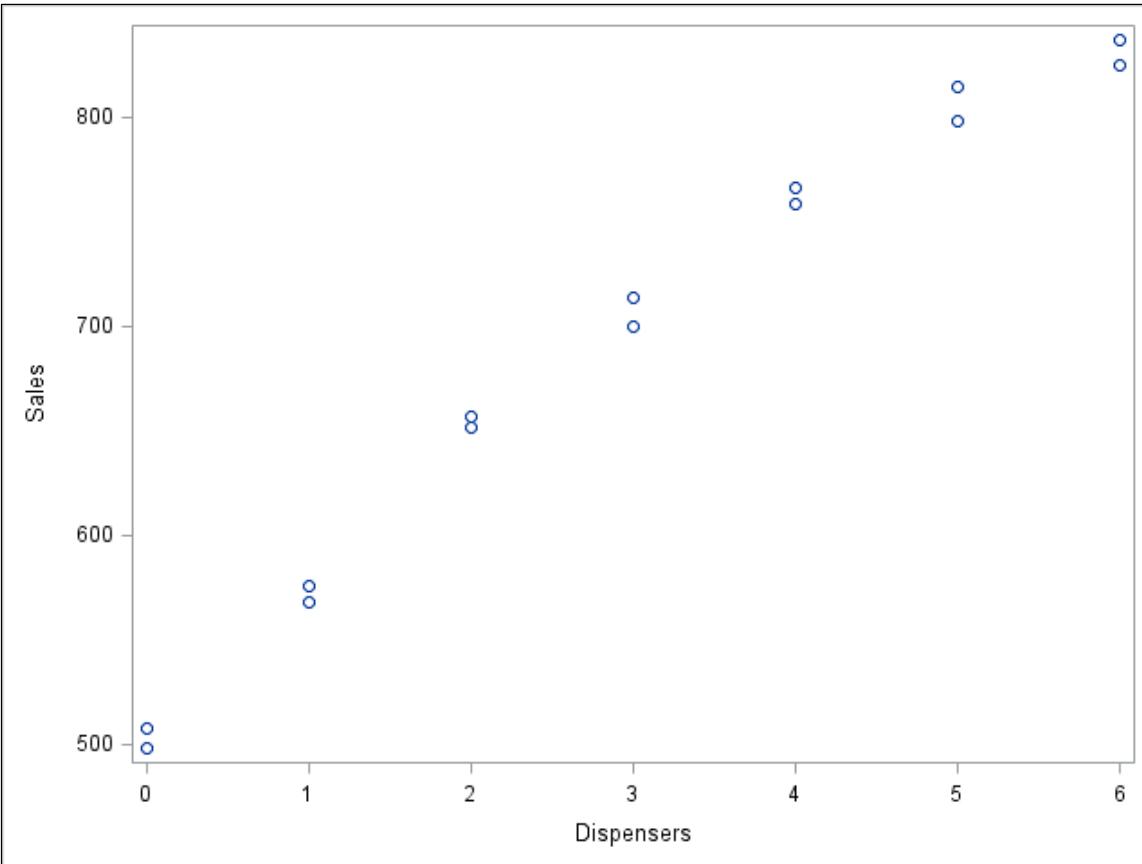
- a. Print the **Cafeteria** data set. What is the range of the number of dispensers for a single cafeteria?

```
proc print data=sp4r.cafeteria;
run;
```

Obs	Cafeteria	Dispensers	Sales
1	1	0	507.9
2	2	0	498.0
3	3	1	568.3
4	4	1	575.5
5	5	2	651.6
6	6	2	657.1
7	7	3	713.8
8	8	3	699.6
9	9	4	758.4
10	10	4	765.7
11	11	5	797.7
12	12	5	814.3
13	13	6	836.8
14	14	6	825.1

- b. Create a scatter plot with **Sales** on the Y axis and **Dispensers** on the X axis. Do the **Sales** appear to be nonlinearly associated with the number of **Dispensers**?

```
proc sgplot data=sp4r.cafeteria;
  scatter x=dispensers y=sales;
run;
```



- c. Use PROC GLMSELECT to create a polynomial regression model. Be sure to use the OUTDESIGN= and PLOT=ALL options in the PROC GLMSELECT statement. Use the EFFECT statement to create a cubic polynomial model with **Dispensers** as the independent variable. Conduct forward selection and use a significance level of 0.05 as the barrier to entry. In addition, use the HIERARCHY=SINGLE option in the MODEL statement to enforce hierarchy. This ensures that higher order terms are in the model only if their corresponding lower order terms are in the model. Which order polynomial does the model select?

```
proc glmselect data=sp4r.cafeteria outdesign=sp4r.des plots=all;
  effect cafe_pol = polynomial(dispensers / degree=3);
  model sales = cafe_pol / selection=forward select=s1 sle=.05
    hierarchy=single;
run;quit;
```

The GLMSELECT Procedure				
Data Set				WORK.CAFETERIA
Dependent Variable				Sales
Selection Method				Forward
Select Criterion				Significance Level
Stop Criterion				Significance Level
Entry Significance Level (SLE)				0.05
Effect Hierarchy Enforced				Single
Number of Observations Read				14
Number of Observations Used				14
Dimensions				
Number of Effects				4
Number of Parameters				4

Forward Selection Summary				
Step	Effect Entered	Number Effects In	F Value	Pr > F
0	Intercept	1	0.00	1.0000

1	Dispensers	2	452.52	<.0001
2	Dispensers^2	3	64.14	<.0001

Selection stopped as the candidate for entry has SLE > 0.05.

Stop Details

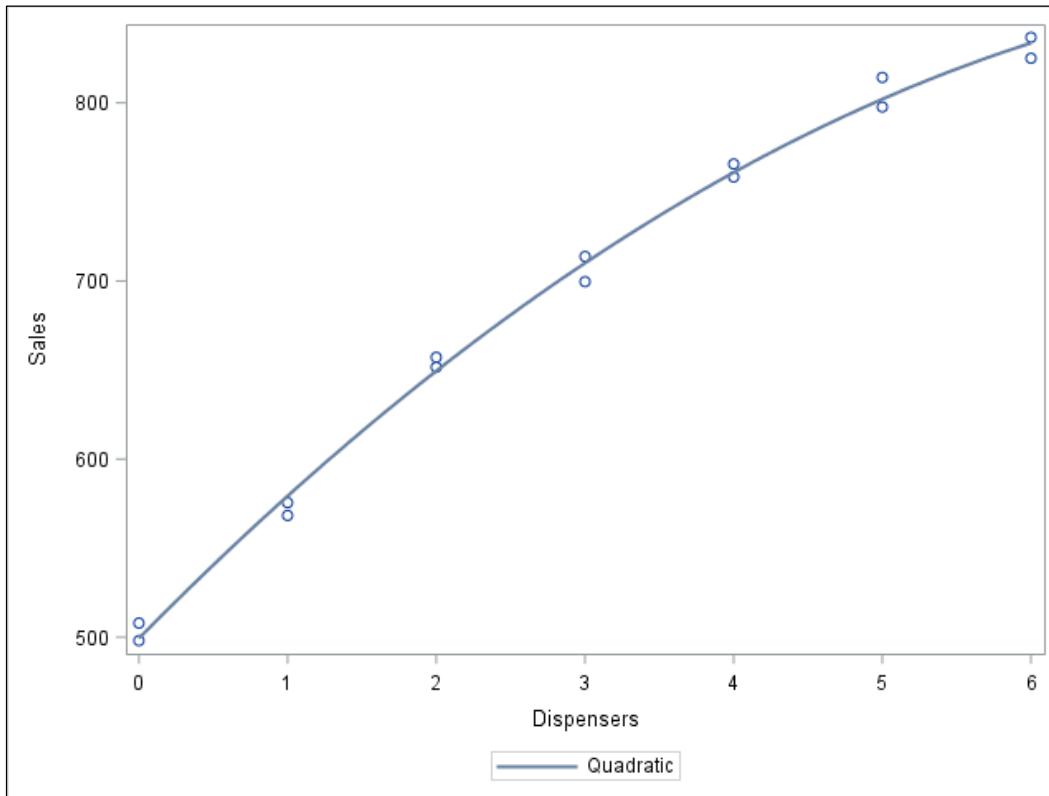
Candidate For	Effect	Candidate Significance	Compare Significance
Entry	Dispensers^3	0.3322	> 0.0500 (SLE)

Effects: Intercept Dispensers Dispensers^2				
Analysis of Variance				
Source	DF	Sum of Squares	Mean Square	F Value
Model	2	177741	88870	1448.87
Error	11	674.71429	61.33766	
Corrected Total	13	178415		
Root MSE				
Dependent Mean				
R-Square				
Adj R-Sq				
AIC				
AICC				
SBC				

Parameter Estimates				
Parameter	DF	Estimate	Standard Error	t Value
Intercept	1	499.371429	4.833914	103.31
Dispensers	1	84.746429	3.773473	22.46
Dispensers^2	1	-4.839286	0.604239	-8.01

- d. Use PROC SGPlot to create a scatter plot and add a line to the degree specified by the final selection model.

```
proc sgplot data=sp4r.cafeteria;
  reg x=dispensers y=sales / degree=2 legendlabel="Quadratic";
run;
```



- e. Use the macro variable (**&_GLSMOD**) and the specified OUTDESIGN= data table to create PROC REG output. Notice that the **GLSMOD** macro variable stores only the significant variables.

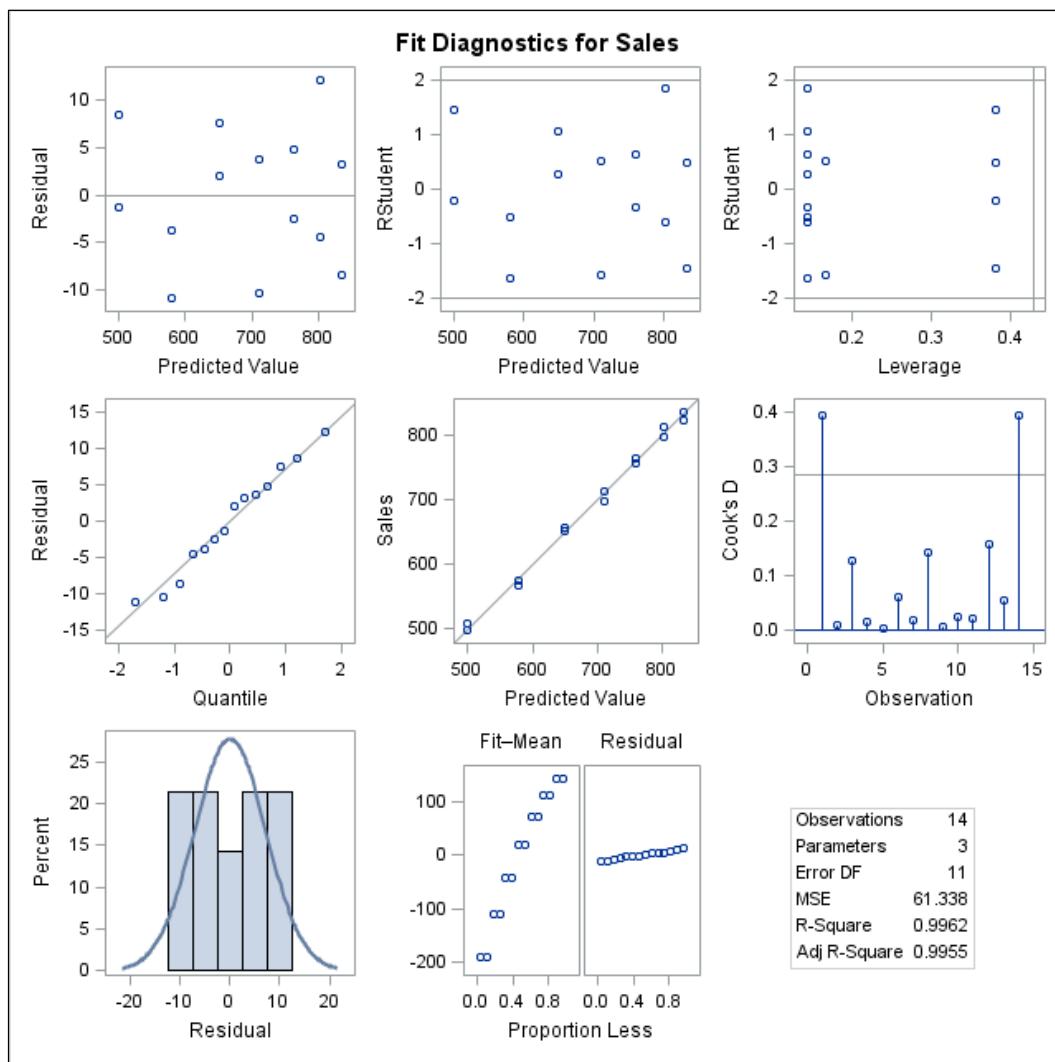
```
proc reg data=sp4r.des;
  model sales = &_glsmod;
run;quit;
```

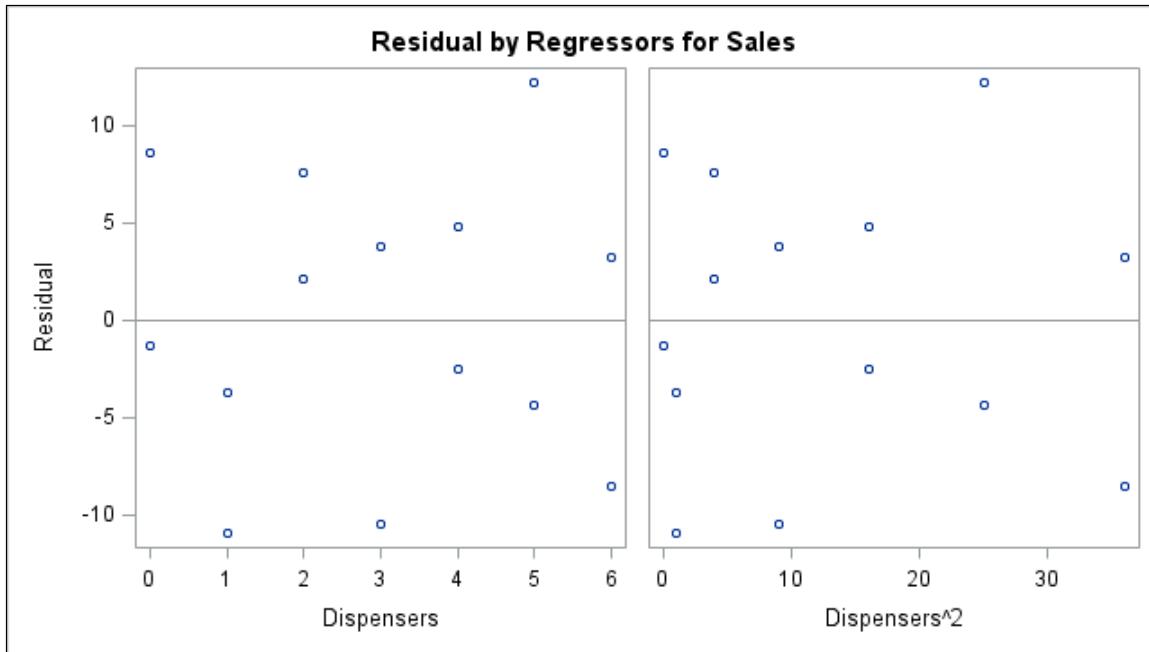
```
The REG Procedure
Model: MODEL1
Dependent Variable: Sales
```

Number of Observations Read	14
Number of Observations Used	14

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	2	177741	88870	1448.87	<.0001
Error	11	674.71429	61.33766		
Corrected Total	13	178415			
		Root MSE	7.83184	R-Square	0.9962
		Dependent Mean	690.70000	Adj R-Sq	0.9955
		Coeff Var	1.13390		

Parameter Estimates					
Variable	Label	DF	Parameter Estimate	Standard Error	Pr > t
Intercept	Intercept	1	499.37143	4.83391	103.31 <.0001
Dispensers	Dispensers	1	84.74643	3.77347	22.46 <.0001
Dispensers_2	Dispensers^2	1	-4.83929	0.60424	-8.01 <.0001





8. Fitting a Logistic Regression Model

The **Safety** data set is created by an insurance company that wants to relate the safety of vehicles to various features. The data set consists of the following variables:

Unsafe 1=Yes, 0=No

Size Size of vehicle (1, 2, or 3)

Weight Weight of Vehicle (1, 2, 3, 4, 5, or 6)

Region Asia or North America

Type Vehicle type (Large, Medium, Small, Sport/Utility, or Sports)

- Use PROC FREQ to create a table of each variable. What percentage of vehicles in the sample are rated as safe?

```
proc freq data=sp4r.safety;
run;
```

The FREQ Procedure					
Unsafe	Frequency	Percent	Cumulative	Cumulative	Percent
			Frequency	Percent	
0	66	68.75	66	68.75	
1	30	31.25	96	100.00	

Size	Frequency	Percent	Cumulative	Cumulative	Percent
			Frequency	Percent	
1	35	36.46	35	36.46	
2	29	30.21	64	66.67	
3	32	33.33	96	100.00	

Weight	Frequency	Percent	Frequency	Percent
1	1	1.04	1	1.04
2	11	11.46	12	12.50
3	53	55.21	65	67.71
4	26	27.08	91	94.79
5	3	3.13	94	97.92
6	2	2.08	96	100.00
Region	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Asia	35	36.46	35	36.46
N America	61	63.54	96	100.00
Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Large	16	16.67	16	16.67
Medium	29	30.21	45	46.88
Small	20	20.83	65	67.71
Sport/Utility	16	16.67	81	84.38
Sports	15	15.63	96	100.00

- b. Use PROC LOGISTIC to model the unsafety of a vehicle with independent variables **Weight**, **Region**, and **Size**. Specify **Region** and **Size** as categorical variables with reference categories *Asia* and *3* respectively. Request only the effect plot for the analysis and use the CLODDS=WALD option in the MODEL statement. Use the ESTIMATE statement to estimate the probability of a vehicle from North America with **Weight**=4 and **Size**=1 as unsafe.

```
proc logistic data=sp4r.safety plots(only)=effect;
  class region(ref='Asia') size(ref='3') / param=ref;
  model unsafe(event='1') = weight region size / clodds=wald;
  estimate 'My Estimate' intercept 1 weight 4 region 1 size 1 0 /
    e alpha=.05 ilink;
run;
```

The LOGISTIC Procedure		
Model Information		
Data Set	WORK.SAFETY	
Response Variable	Unsafe	
Number of Response Levels	2	
Model	binary logit	
Optimization Technique	Fisher's scoring	
Number of Observations Read	96	
Number of Observations Used	96	
Response Profile		
Ordered Value	Unsafe	Total Frequency
1	0	66
2	1	30

Probability modeled is Unsafe=1.

Class Level Information

Class	Value	Design Variables	
Region	Asia	0	1
	N America		
Size	1	1	0
	2	0	1
	3	0	0

Model Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Intercept Only	Intercept and Covariates
AIC	121.249	94.004
SC	123.813	106.826
-2 Log L	119.249	84.004

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	35.2441	4	<.0001
Score	32.8219	4	<.0001
Wald	23.9864	4	<.0001

Type 3 Analysis of Effects

Effect	DF	Chi-Square	Pr > ChiSq
Weight	1	2.1176	0.1456
Region	1	0.4506	0.5020
Size	2	15.3370	0.0005

Analysis of Maximum Likelihood Estimates					
Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	0.0500	1.8008	0.0008	0.9778
Weight	1	-0.6678	0.4589	2.1176	0.1456
Region N America	1	-0.3775	0.5624	0.4506	0.5020
Size 1	1	2.6783	0.8810	9.2422	0.0024
Size 2	1	0.6582	0.9231	0.5085	0.4758

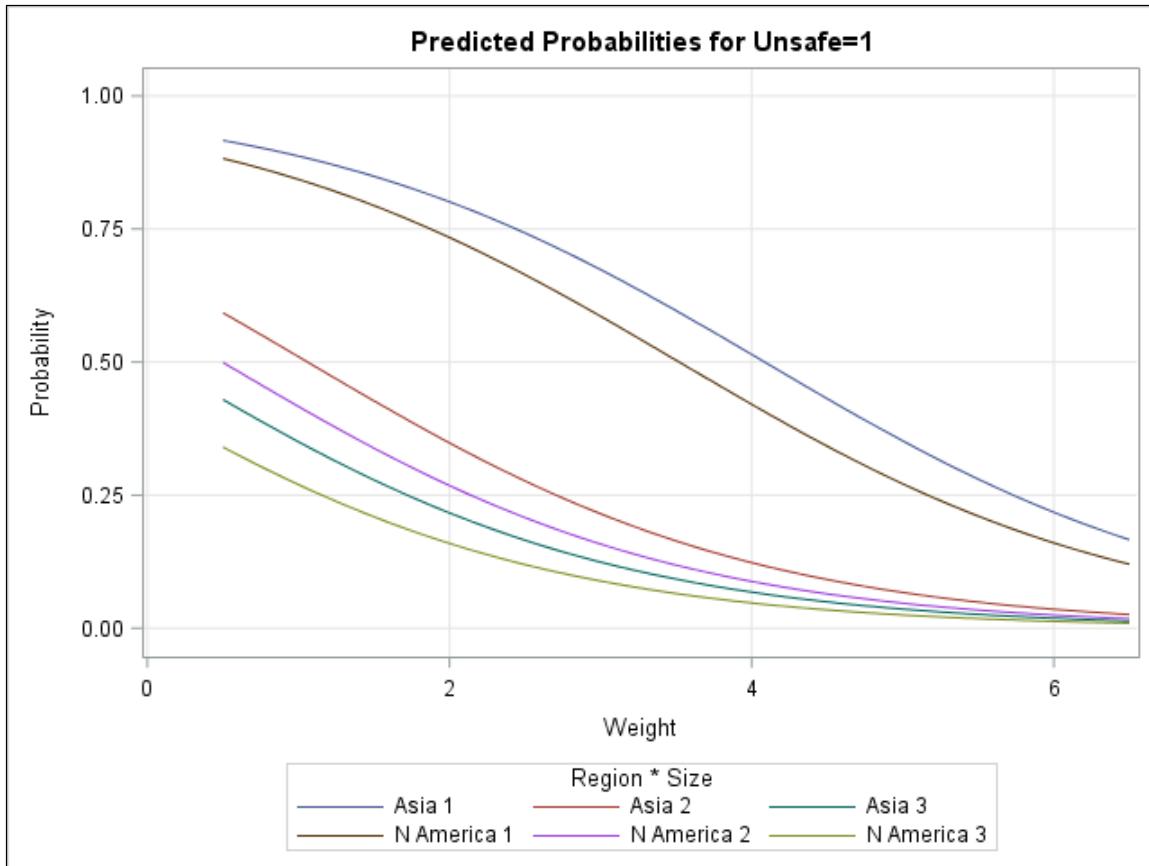
Association of Predicted Probabilities and Observed Responses					
Percent Concordant	81.9	Somers' D	0.696		
Percent Discordant	12.3	Gamma	0.739		
Percent Tied	5.8	Tau-a	0.302		
Pairs	1980	c	0.848		

Odds Ratio Estimates and Wald Confidence Intervals					
Effect	Unit	Estimate	95% Confidence Limits		
Weight	1.0000	0.513	0.209	1.261	
Region N America vs Asia	1.0000	0.686	0.228	2.064	
Size 1 vs 3	1.0000	14.560	2.590	81.857	
Size 2 vs 3	1.0000	1.931	0.316	11.793	

Estimate Coefficients					
Parameter	Region	Size	Row1		
Intercept: Unsafe=0				1	
Weight				4	
Region N America	N America			1	
Size 1		1		1	
Size 2		2			

Estimate									
Label	Estimate	Error	z Value	Pr > z	Alpha	Lower	Upper	Mean	Error of Mean
My Estimate	-0.3204	0.6916	-0.46	0.6432	0.05	-1.6759	1.0352	0.4206	0.1685

Estimate									
Label	Mean	Lower Mean	Upper Mean						
My Estimate	0.1576	0.1576	0.7379						



- c. Because the significance of the parameter estimates were weak for the model above, conduct backward selection with a significance level to stay in the model of 0.05. Use ODS SELECT to request only the ModelBuildingSummary, ModelAnova, and ParameterEstimates values of the final model. Which variables were removed from the model?

```
ods select modelbuildingsummary modelanova parameterestimates;
proc logistic data=sp4r.safety;
  class region(ref='Asia') size(ref='3') / param=ref;
  model unsafe(event='1') = weight region size /
    selection=backward sls=.05 clodds=wald;
run;
```

The LOGISTIC Procedure					
Summary of Backward Elimination					
Step	Effect Removed	DF	Number In	Wald Chi-Square	Pr > ChiSq
1	Region	1	2	0.4506	0.5020
2	Weight	1	1	2.1565	0.1420

Type 3 Analysis of Effects			
Effect	DF	Chi-Square	Wald ChiSq

Size	2	24.2875	<.0001		
Analysis of Maximum Likelihood Estimates					
Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-2.7080	0.7303	13.7505	0.0002
Size	1	3.3585	0.8125	17.0880	<.0001
Size	2	1.1393	0.8803	1.6751	0.1956

9. Fitting a Generalized Linear Model

A Survey was undertaken to examine which factors are related to ear infections among swimmers. The response variable is the number of self-diagnosed ear infections reported by the participant. The data are stored in the **EarInfection** data set. The data set consists of the following variables:

Infections	Number of self-diagnosed ear infections
Swimmer	Frequent or Occasional swimmer
Location	Typical swimming location (NonBeach or Beach)
Age	Age in years
Gender	Gender of swimmer (Male or Female)

 The data were obtained with permission from the OZDATA website. This website is a collection of data sets and is maintained in Australia.

- a. Create a frequency table for the variables **Swimmer**, **Location**, **Age**, **Gender**, and **Infections**. What is the range of the number of infections for swimmers in this sample?

```
proc freq data=sp4r.earinfection;
  tables swimmer location age gender infections;
run;
```

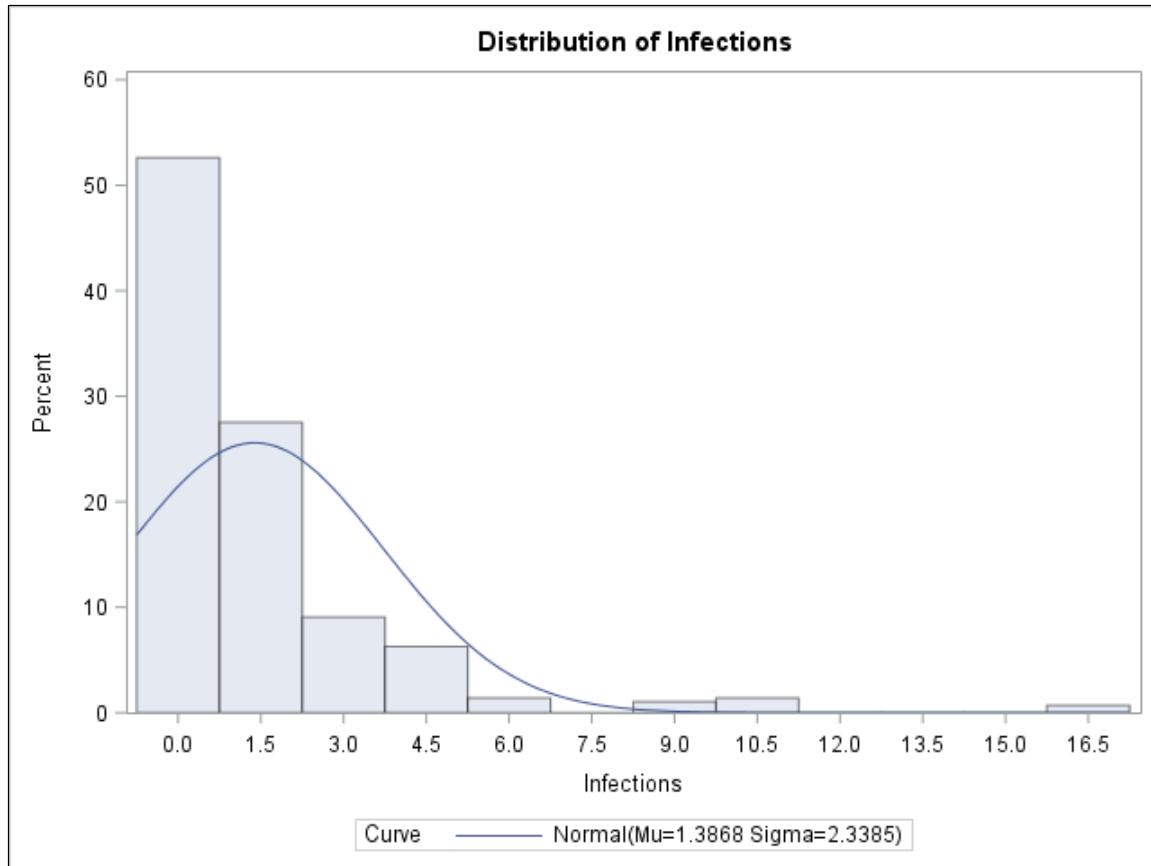
The FREQ Procedure				
			Cumulative Frequency	Cumulative Percent
Swimmer	Frequency	Percent		
Freq	143	49.83	143	49.83
Occas	144	50.17	287	100.00
Location	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Beach	147	51.22	147	51.22
NonBeach	140	48.78	287	100.00
Age	Frequency	Percent	Cumulative Frequency	Cumulative Percent
15	28	9.76	28	9.76
16	26	9.06	54	18.82
17	28	9.76	82	28.57
18	32	11.15	114	39.72

19	26	9.06	140	48.78
20	16	5.57	156	54.36
21	15	5.23	171	59.58
22	16	5.57	187	65.16
23	16	5.57	203	70.73
24	16	5.57	219	76.31
25	12	4.18	231	80.49
26	14	4.88	245	85.37
27	14	4.88	259	90.24
28	15	5.23	274	95.47
29	13	4.53	287	100.00
Gender	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Female	99	34.49	99	34.49
Male	188	65.51	287	100.00
Infections	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	151	52.61	151	52.61
1	40	13.94	191	66.55
2	39	13.59	230	80.14
3	26	9.06	256	89.20
4	13	4.53	269	93.73
5	5	1.74	274	95.47
6	4	1.39	278	96.86
9	3	1.05	281	97.91
10	3	1.05	284	98.95
11	1	0.35	285	99.30
16	1	0.35	286	99.65
17	1	0.35	287	100.00

- b. Request only the BasicMeasures table and a histogram with a normal density estimate from PROC UNIVARIATE for the **Infections** variable. Does this variable appear to be normally distributed?

```
ods select basicmeasures histogram;
proc univariate data=sp4r.earinfection;
  var infections;
  histogram infections / normal;
run;
```

The UNIVARIATE Procedure			
Variable: Infections			
Basic Statistical Measures			
Location		Variability	
Mean	1.386760	Std Deviation	2.33854
Median	0.000000	Variance	5.46878
Mode	0.000000	Range	17.00000
		Interquartile Range	2.00000



- c. Create a Poisson regression model with **Infections** as the dependent variable and **Swimmer**, **Location**, **Age**, and **Gender** as the independent variables. For the variables **Swimmer**, **Location**, and **Gender** use the reference categories Occas, NonBeach, and Female respectively. Be sure to use the STORE statement to predict the number of Infections using PROC PLM. Which variables are statistically significant?

```
proc genmod data=sp4r.earinfection;
  class swimmer(ref='Occas') location(ref='NonBeach')
    gender(ref='Female') / param=ref;
  model infections = swimmer location age gender /
    dist=poisson type3;
  store mymod;
run;
```

The GENMOD Procedure	
Model Information	
Data Set	WORK.EARINFECTION
Distribution	Poisson
Link Function	Log
Dependent Variable	Infections
Number of Observations Read	287
Number of Observations Used	287
Class Level Information	

Design Variables			
Class	Value		
Swimmer	Freq	1	
	Occas	0	
Location	Beach	1	
	NonBeach	0	
Gender	Female	0	
	Male	1	
Criteria For Assessing Goodness Of Fit			
Criterion	DF	Value	Value/DF
Deviance	282	760.0060	2.6951
Scaled Deviance	282	760.0060	2.6951
Pearson Chi-Square	282	963.5838	3.4170
Scaled Pearson X2	282	963.5838	3.4170
Log Likelihood		-235.6148	
Full Log Likelihood		-566.2004	
AIC (smaller is better)		1142.4008	
AICC (smaller is better)		1142.6143	
BIC (smaller is better)		1160.6982	

Algorithm converged.

Analysis Of Maximum Likelihood Parameter Estimates							
Parameter	DF	Estimate	Standard Error	Wald 95% Confidence Limits		Chi-Square	Pr > ChiSq
				Lower	Upper		
Intercept	1	1.3586	0.2736	0.8224	1.8948	24.66	<.0001
Swimmer	Freq	1	-0.6086	0.1050	-0.8145	-0.4028	33.59
Location	Beach	1	-0.4896	0.1048	-0.6951	-0.2841	21.81
Age		1	-0.0261	0.0122	-0.0500	-0.0021	4.55
Gender	Male	1	-0.0294	0.1092	-0.2433	0.1846	0.07
Scale	0	1.0000	0.0000	1.0000	1.0000		0.7878

NOTE: The scale parameter was held fixed.

LR Statistics For Type 3 Analysis

Source	DF	Chi-Square	
			Pr > ChiSq
Swimmer	1	35.16	<.0001
Location	1	22.35	<.0001
Age	1	4.64	0.0312
Gender	1	0.07	0.7881

- d. Create a new data set with the following observations:

Swimmer	Location	Age	Gender
Freq	NonBeach	25	Female

Occas

Beach

15

Male

```

data sp4r.newdata_inf;
  input Swimmer $ Location $ Age Gender $;
  datalines;
Freq NonBeach 25 Female
Occas Beach 15 Male
;run;

```

- e. Use PROC PLM to score the new data set and predict the number of **Infections** on the original data scale. Finally, print the predicted values from the PROC PLM output data set.

```

proc plm restore=mymod;
  score data=sp4r.newdata_inf out=sp4r.scores / ilink;
run;

```

The PLM Procedure

Store Information

Item Store	WORK.EARPRED
Data Set Created From	WORK.EARINFECTION
Created By	PROC Genmod
Date Created	13OCT15:09:52:16
Response Variable	Infections
Link Function	Log
Distribution	Poisson
Class Variables	Swimmer Location Gender
Model Effects	Intercept Swimmer Location Age Gender

```

proc print data=sp4r.scores;
run;

```

Obs	Swimmer	Location	Age	Gender	Predicted
1	Freq	NonBeach	25	Female	1.10338
2	Occas	Beach	15	Male	1.56621

10. Fitting a Two-Way Mixed Model

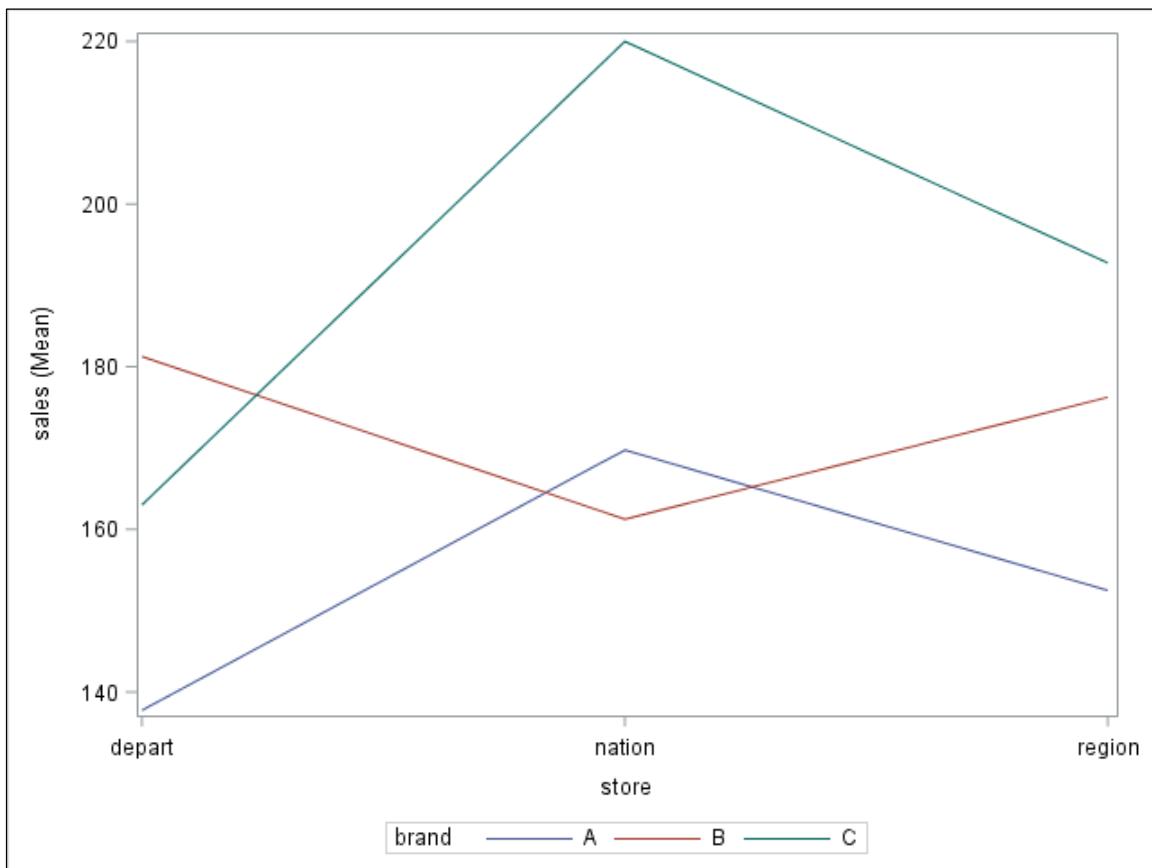
- a. Use PROC SGLOT to plot the average value of **sales** (y) versus **store** (x) for each **brand** (group). What do you conclude from the graph?

```

proc sgplot data=sp4r.washer;
  vline store / group=brand stat=mean response=sales;
run;

```

PROC SGLOT Output



It seems that **brand C** consistently has better sales than **brand A** across all stores. The sales averaged across all brands seem to be pretty consistent across stores. There seems to be some **brand*store** interaction.

- b. Is **brand** a fixed effect or a random effect? How would you classify **store** and **brand*store**?
The variable brand is a fixed effect, whereas store and brand*store are random effects.
- c. Analyze this design using the MIXED procedure and use the REML method. What are the estimates of variance components? Is **brand** a significant factor?

```
proc mixed data=sp4r.washer method=reml;
  class brand store;
  model sales=brand / ddfm=kr2;
  random store brand*store;
run;
```

Partial PROC MIXED Output

Covariance Parameter Estimates	
Cov Parm	Estimate
store	3.4931
brand*store	339.57
Residual	197.91

The estimated variance components are 3.49 for store, 339.57 for brand*store interaction, and 197.91 for Residual.

Type 3 Tests of Fixed Effects

Effect	Num DF	Den DF	F Value	Pr > F
brand	2	4	2.87	0.1687

The variable **brand** does not appear to have a significant effect on the sales across the entire population of types of stores.

- d. Add an ESTIMATE statement to compare the average monthly sales of **brand A** and **brand B**.

```
proc mixed data=sp4r.washer;
  class brand store;
  model sales=brand / ddfm=kr2;
  random store brand*store;
  estimate 'brand A vs brand B' brand 1 -1;
run;
```

Partial PROC MIXED Output

Estimates

Label	Estimate	Standard Error	DF	t Value	Pr > t
brand A vs brand B	-19.5833	16.1048	4	-1.22	0.2908

The average monthly sales for **brand A** is 19.58 lower than those for **brand B**, and this difference is not statistically significant (*p*-value=0.2908).

- e. Use the Expected Mean Squares table to test the variance components.

```
proc mixed data=sp4r.washer method=type3;
  class brand store;
  model sales = brand;
  random store brand*store;
  estimate 'brand A vs B' brand 1 -1;
run;
```

Partial PROC MIXED Output

Type 3 Analysis of Variance

Source	DF	Sum of Squares	Mean Square	Expected Mean Square
brand	2	8932.722222	4466.361111	Var(Residual) + 4 Var(brand*store) + Q(brand)
store	2	3196.222222	1598.111111	Var(Residual) + 4 Var(brand*store) + 12 Var(store)
brand*store	4	6224.777778	1556.194444	Var(Residual) + 4 Var(brand*store)
Residual	27	5343.500000	197.907407	Var(Residual)
Source	Error Term		Error	
			DF	F Value
				Pr > F

brand	MS(brand*store)	4	2.87	0.1687
store	MS(brand*store)	4	1.03	0.4366
brand*store	MS(Residual)	27	7.86	0.0002
Residual

Only the Brand*Store interaction variance component is significant at the 0.05 level of significance.

End of Solutions

Solutions to Student Activities (Polls/Quizzes)

6.01 Multiple Answer Poll – Correct Answers

Which statements are correct? (Select all that apply.)

- a. The PLM procedure uses the model specified by the STORE statement.
- b. PROC REG uses a CLASS statement to specify categorical variables.
- c. The PLM procedure SCORE statement keywords provide interval output.
- d. The PLM procedure scores new data sets.

16

6.02 Multiple Choice Poll – Correct Answer

Choose the correct statements.

- a. The CLASS statement is equivalent to the as.factor() function in R.
- b. The MEANS statement provides the same default output as PROC MEANS.
- c. The ADJUST= option in the LSMEANS statements is used to adjust for multiple comparisons.
- d. The SOLUTION option in the MODEL statement is used to display parameter estimates.

30

6.03 Poll – Correct Answer

The GLMSELECT procedure can be used to create a regression model, a polynomial regression model, an ANOVA model, an ANCOVA model, and to conduct stepwise model selection.

- True
 False

48

6.04 Multiple Choice Poll – Correct Answer

Consider a logistic regression model where the binary response is whether a person is a dog owner. You sample 140 people and find that 95 people are dog owners. How many total concordant and discordant pairs are considered?

- a. 140
 b. $95 \times 45 = 4275$
c. $(95 \times 45) / 140 = 30.53$
d. $95 \times 45 \times 140 = 598500$

65

Correct Answer: B

The total number of pairs is the number of successes times the number of failures (95×45).

6.05 Poll – Correct Answer

Logistic regression can be conducted in both PROC LOGISTIC and PROC GENMOD.

- True
 False

Chapter 7 Interactive Matrix Language (IML)

7.1 The Basics (Self-Study).....	7-3
Demonstration: Working in Interactive Mode (SAS Studio)	7-11
Demonstration: Basic Matrix Operations	7-12
Exercises	7-15
7.2 Modules and Subroutines	7-17
Demonstration: Simple Linear Regression	7-23
Demonstration: Navigating the SAS/IML Documentation	7-30
Demonstration: Creating Functions and Subroutines	7-43
7.3 Calling SAS Data Sets and Procedures.....	7-45
Demonstration: Calling SAS Data Sets from SAS/IML	7-52
Demonstration: Submitting SAS Procedures	7-59
7.4 Simulations	7-61
Demonstration: The Monty Hall Problem	7-68
Demonstration: Sampling Distribution – Method 1	7-72
Demonstration: Sampling Distribution – Method 2	7-74
Demonstration: Sampling Distribution – Method 3	7-77
Exercises	7-79
7.5 Solutions	7-86
Solutions to Exercises	7-86
Solutions to Student Activities (Polls/Quizzes)	7-99

7.1 The Basics (Self-Study)

Objectives

- Create matrix literals.
- Use basic matrix and comparison operators.
- Use subscript reduction operators.
- Explain implicit looping.

3

Motivation

Use matrix algebra to customize statistical analyses.

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k + \varepsilon$$



PROC Step



$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

4

Duplicating the R Script

```

Source on Save | Magnifying Glass | Print | Copy | Paste | Source
#Create matrices
x = matrix(c(1,2,3,4),nrow=2,ncol=2,byrow=T)
y = matrix(c(5,6,7,8),nrow=2,ncol=2,byrow=T)

#Elementwise operations
x ^ 2
x * y

#Matrix operations
x %*% y

#Accessing elements
x[1,]
y[1,1:2]

#Reduction operators
max(x)
sum(x)
colMeans(x)

```

5

IML Procedure

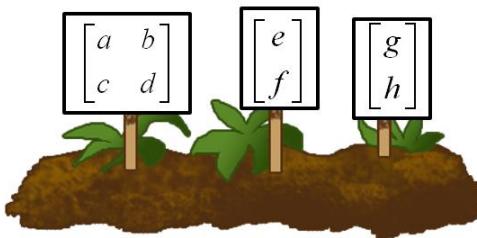
General form of a PROC IML step:

```

PROC IML;
  IML syntax
QUIT;

```

To save space, the examples in this class tend to leave out the PROC IML; and QUIT; lines.



6

PROC IML can be used interactively or in batch mode. Using IML in batch mode entails submitting the PROC IML call, a set of IML statements, and finally a QUIT statement. PROC IML does not require a RUN statement.

To use IML in interactive mode, submit the PROC IML call. IML statements can then be submitted one at a time or in groups. When IML is no longer needed, submit the QUIT statement to exit IML.

Brackets, Braces, and Parentheses

Brackets, braces, and parentheses have distinct uses in SAS/IML and are not interchangeable.

- Brackets {} are used for making a matrix from literal values.
- Braces [] are used for referring to matrix subscripts and for specifying options in IML statements.
- Parentheses () are used for SAS functions and for controlling the order of operations.

7

Creating Matrices with Matrix Literals

Use a comma to begin a new matrix row.

```
x = matrix(c(1,2,3,4,5,6), nrow=2, ncol=3, byrow=T)
```

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Create a character matrix.

```
x = matrix(c("Jordan", "Baker", "Man"), nrow=3, ncol=1, byrow=T)
```

$$X = \begin{bmatrix} Jordan \\ Baker \\ Man \end{bmatrix}$$

8

SAS/IML matrices must contain either character or numeric elements. A matrix cannot contain both types of elements. Numeric elements are stored in double-precision floating point-format using eight bytes. Elements of character matrices can be from 1 to 32,767 bytes long. Matrices are referenced by valid SAS names. Names can be from 1 to 32 characters long, beginning with a letter or an underscore and continuing with any combination of letters, underscores, or digits.

The length of each element in a character matrix is determined dynamically to be the length of the longest element. For example, if you assign the value *dog* to element 1 of a matrix and the value *horse* to element 2, then the first element is five bytes long, and the last two bytes are blank characters. If you later change element 2 to *cat*, the length of elements remains five unless a longer element was assigned. The LENGTH statement can be used to determine the length of each element in a character matrix.

If you assigned character values to a matrix, and then you assign an element, the value 2 not enclosed in quotation marks, the element contains the character '2', not the floating point numeric representation of 2, because matrices cannot be of mixed type.

PRINT Statement

SAS does not have a command line. Therefore, use a PRINT statement to view IML matrices.

```
x = matrix(c(1,2,3,4,5,6),nrow=2,ncol=3,byrow=T)
x
```

```
x = {1 2 3,
      4 5 6}
print x;
```

```
PRINT matrices <expression> <"message"> <pointers> <[options]>;
```

Use parentheses to print a matrix without an assignment statement.

```
print ( {1 2 3,
          4 5 6} );
```

9

General form of the PRINT statement:

matrices names the matrices to be printed. (More than one can be specified in a single PRINT statement.)

expression resolves to a matrix to be printed.

message specifies the text to print.

pointers include a comma to skip a line and a slash to start a new page.

options enable the addition of labels and formats.

Here are some PRINT statement options:

FORMAT= specifies a valid SAS or user-defined format to use in printing the values of the matrix.

COLNAME=*matrix* specifies the name of a character matrix whose first *ncol* elements are to be used for the column labels of the matrix to be printed.

ROWNAME=*matrix* specifies the name of a character matrix whose first *nrow* elements are to be used for the row labels of the matrix to be printed.

LABEL= specifies a label for the matrix to be printed.

7.01 Multiple Answer Poll

Suppose you want to print your salary for the week. Assume that you worked 40 hours and earn \$9.35 per hour. Which of the following show the correct syntax for printing your salary? (Select all that apply.)

- You do not need brackets to assign a scalar.
- a. `Y=40*9.35; Print Y;`
- b. `40*9.35;`
- c. `Print(40*9.35);`
- d. `Print 40*9.35;`

10

Similarities to R

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

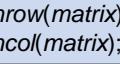
- Accessing matrix elements:

`x[2,1];`  `[4]`
`matrix-name[row,column];`

- Accessing columns or rows:

`x[,1];`  `[4 5 6]`
`matrix[,row];` `matrix[,column];`

- Finding dimensions:

`D1=nrow(x);`  `[2]`
`D2=ncol(x);`  `[3]`
`nrow(matrix);` `ncol(matrix);`

12

SAS/IML does not have a `dim()` function.

Creating a Sequence

- The Index operator is identical to R.

`x = 1:5;` [1 2 3 4 5]
 value1:value2;

- Use a DO function to create a sequence.

Source on Save seq(from=2,to=10,by=2)

`x = do(2,10,2);`
 vector = DO(start,stop,increment);

[2 4 6 8 10]

13

Basic Operators

Description	SAS Operator	R Operator
Elementwise	+, -, #, /, ##	+, -, *, /, ^
Matrix Multiplication	*	%*%
Matrix Exponentiation	**	
Transpose	t() or `	t()
Horizontal Concatenation		rbind()
Vertical Concatenation	//	cbind()



The SAS matrix multiplication operator is the same as the R elementwise multiplication operator.

14

Comparison Operators

Description	SAS Operator	R Operator
Less than	<	<
Less than or equal to	<=	<=
Equal to	=	==
Not equal to	^=	!=
Greater than	>	>
Greater than or equal to	>=	>=

Example: $\mathbf{z} = \{2 \ 7, \ 3 \ 5\} > \{4 \ 6, \ 5 \ 8\};$

$$\begin{bmatrix} 2 & 7 \\ 3 & 5 \end{bmatrix} > \begin{bmatrix} 4 & 6 \\ 5 & 8 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

15

Comparison operators perform elementwise comparisons and produce a new matrix that contains only 0s and 1s. If an element comparison is true, the corresponding element of the new matrix is 1. If the comparison is not true, the corresponding element is 0. Unlike in Base SAS, you cannot use the mnemonic equivalents (GT, LT, GE, LE, NE, EQ).

Implicit Looping

Some matrix operations can be performed on matrices that are not conformable to the operation!

- Matrix and scalar:

$$\begin{bmatrix} 2 & 6 \\ 5 & 9 \end{bmatrix} + 3 = \begin{bmatrix} 5 & 9 \\ 8 & 12 \end{bmatrix}$$

- Matrix and row vector:

$$\begin{bmatrix} 2 & 6 \\ 5 & 9 \end{bmatrix} - [5 \ 2] = \begin{bmatrix} -3 & 4 \\ 0 & 7 \end{bmatrix}$$

- Matrix and column vector:

$$\begin{bmatrix} 2 & 6 \\ 5 & 9 \end{bmatrix} / [2 \ 4] = \begin{bmatrix} 1 & 3 \\ 1.25 & 2.25 \end{bmatrix}$$

16

If \mathbf{X} is an m -by- n matrix, and \mathbf{Y} is a 1-by- n row vector, then the expression $\mathbf{X}+\mathbf{Y}$ evaluates to the addition of \mathbf{Y} to each row of \mathbf{X} . This change was introduced to reduce the need for explicit loops and increase the efficiency of this type of calculation.

Subscript Reduction Operators

Operator	Description
+	Sum
#	Product
<>	Maximum
<<	Minimum
<:>	Index of maximum
>:<	Index of minimum
:	Mean
##	Sum of squares

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 9 \\ 7 & 8 & 4 \end{bmatrix}$$

- $Y=X[+,]$ produces $Y=[14 \ 15 \ 16]$
- $Y=X[, <>]$ produces $Y=[3, 9, 8]$
- $Y=X[\#,]$ produces $Y=[42 \ 80 \ 108]$
- $Y=X[:,]$ produces $Y=[14/3 \ 5 \ 16/3]$

17

To help you decide whether to specify an operator in the row or column dimension, remember that if a dimension's subscript is missing, that dimension remains unchanged in the new matrix. In the first example above, the subscript for the column dimension is missing, so the resulting matrix has the same number of columns, three.

You can use reduction operators to reduce either rows or columns or both. When both rows and columns are reduced, row reduction is done first.

7.02 Multiple Choice Poll

Let X be an m -by- n matrix. How would you use a SAS reduction operator to reproduce the **rowMeans()** and **min()** functions in R?

- $X[:,]$ and $X[><,]$
- $X[:,]$ and $X(>:<, >:<)$
- $X{:, } and $X[><, ><]$$
- $X[:,]$ and $X[><, ><]$

18

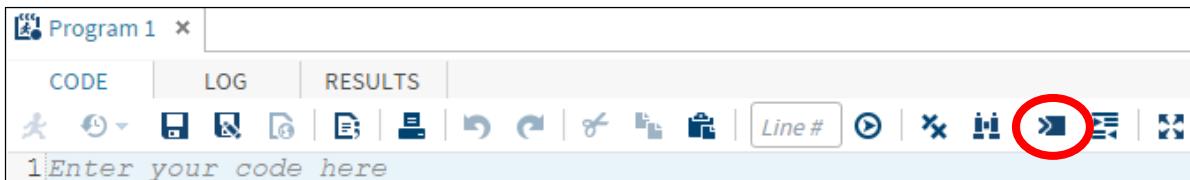


Working in Interactive Mode (SAS Studio)

Some SAS procedures are interactive, which means that they remain active until you submit a QUIT statement or until you submit a new PROC or DATA step. In SAS Studio, you can use the code editor to run these procedures as well as other SAS procedures in interactive mode.

By using interactive mode, you can run selected lines of code from your SAS program and use the results to determine your next step.

1. Click the **Go Interactive** button from the SAS Studio Code Editor tools.



2. Invoke the IML procedure.

```
proc iml;
```

Now all subsequent code is executed in the matrix language until a QUIT statement is submitted.

3. Create a vector of **grades** and submit a PRINT statement.

```
grades={95,91,73,88,96};  
print grades;
```

grades

95
91
73
88
96

4. Find the mean of the **grades** vector and print it.

```
mymean=grades[ : ];  
print mymean;
```

mymean

88.6

- The results and the log do not clear when interactive mode is turned on and PROC IML needs to be run a single time only.

5. Use a QUIT statement to end the IML procedure.

```
quit;
```

End of Demonstration



Basic Matrix Operations

SP4R07d01.sas

- To practice some of the basic matrix operations in SAS/IML, consider an example program used to generate a weekly expense report. The example program follows.

```
proc iml;
  items      ={'Groceries','Utilities','Rent','Car Expenses',
               'Fun Money','Personal Expenses'};
  weeks      ={'Week 1','Week 2','Week 3','Week 4'};
  amounts    ={ 96 78 82 93,
               61 77 62 68,
               300 300 300 300,
               25 27 98 18,
               55 34 16 53,
               110 85 96 118};
  weeklyIncome={900 850 1050 950};
  print items amounts weeklyIncome;
  print items, amounts, weeklyIncome;
```

Assignment statements in PROC IML are used to define a numerical expense matrix, a vector of weekly incomes, and character label matrices.

The statement **print items amounts weeklyIncome;** prints the specified matrices side by side.

PROC IML Output

items	amounts					weeklyIncome							
Groceries	96	78	82	93		900	850	1050	950				
Utilities	61	77	62	68									
Rent	300	300	300	300									
Car Expenses	25	27	98	18									
Fun Money	55	34	16	53									
Personal Expenses	110	85	96	118									

The statement **print items, amounts, weeklyIncome;** prints the specified matrices independently.

PROC IML Output

items
Groceries
Utilities
Rent
Car Expenses
Fun Money
Personal Expenses

amounts			
96	78	82	93
61	77	62	68
300	300	300	300
25	27	98	18
55	34	16	53
110	85	96	118
weeklyIncome			
900	850	1050	950

2. Compute a vector containing weekly expenses. Turn off the automatic printing of matrix names. Print the matrix using the PRINT statement with an appropriate title and column labels. Change the format to reflect that the numbers refer to dollar amounts.

```
reset noname;
weeklyTotals=amounts[+,];
print "Total Expenses for Each Week",
      weeklyTotals [colname=weeks format=dollar8.2];
```

The statement `weeklyTotals=amounts[+,]`; uses a reduction operator to create a row vector named `weeklyTotals` that contains the column totals of `amounts` that is the weekly total spent.

PROC IML Output

Total Expenses for Each Week			
Week 1	Week 2	Week 3	Week 4
\$647.00	\$601.00	\$654.00	\$650.00

3. Calculate the total expenses for each item across the entire month. Print the matrix using the PRINT statement with an appropriate title and row labels. Change the format to reflect that the numbers refer to dollar amounts.

```
itemTotals=amounts[,+];
print "Total Expenses for Each Item",
      itemTotals[rowname=items format=dollar8.2];
```

The statement `itemTotals=amounts[,+]`; uses a reduction operator to create a column vector named `itemTotals` that contains the row totals of `amounts` that is the monthly total spent on each item.

PROC IML Output

Total Expenses for Each Item	
Groceries	\$349.00
Utilities	\$268.00
Rent	\$1200.00
Car Expenses	\$168.00
Fun Money	\$158.00
Personal Expenses	\$409.00

4. Calculate the percentage of weekly income spent on each item for each week. That is, use an implicit loop to create a new matrix, `weeklyPercentage`, equal to `amounts` divided by `weeklyTotals`. Print the matrix using the PRINT statement with an appropriate title, row labels, and column labels. Change the format to reflect that the numbers refer to percentages.

```
weeklyPercentage=amounts/weeklyIncome;
print "Percentage of income spent on each item, by week",
      weeklyPercentage[rowname=items colname=weeks format=percent7.2];
```

Because **amounts** is a 6 x 4 matrix and **weeklyIncome** is a 1 x 4 row vector, the statement **weeklyPercentage=amounts/weeklyIncome;** tells IML to use an implicit loop in which each row of **amounts** is divided by **weeklyIncome**.

Percentage of income spent on each item, by week				
	Week 1	Week 2	Week 3	Week 4
Groceries	10.7%	9.18%	7.81%	9.79%
Utilities	6.78%	9.06%	5.90%	7.16%
Rent	33.3%	35.3%	28.6%	31.6%
Car Expenses	2.78%	3.18%	9.33%	1.89%
Fun Money	6.11%	4.00%	1.52%	5.58%
Personal Expenses	12.2%	10.0%	9.14%	12.4%

5. Calculate **monthlyIncome** as the sum of the elements in **weeklyIncome**. Calculate **itemProportion** as the proportion of monthly income spent on each item. Print **itemProportion** with a title and appropriate row names, and change the format to represent the numbers as percentages.

```
monthlyIncome =weeklyIncome[+];
itemProportion =itemTotals/monthlyIncome;
print "Each Item as a Percentage of Monthly Income",
            itemProportion[rowname=items format=percent7.2];
quit;
```

The statement **monthlyIncome=weeklyIncome [+]** ; uses a subscript reduction operator to create a scalar that is the sum of all of the elements of the matrix **weeklyIncome**.

The statement **itemProportion=itemTotals/monthlyIncome** ; uses an implicit loop to divide each element in **itemTotals** by the scalar **monthlyIncome** and place the result in **itemProportion**.

The FORMAT= option in the PRINT statement specifies the PERCENT7.2 format to display the proportions in the vector **itemProportion** as percentages.

PROC IML Output

Each Item as a Percentage of Monthly Income	
Groceries	9.31%
Utilities	7.15%
Rent	32.0%
Car Expenses	4.48%
Fun Money	4.21%
Personal Expenses	10.9%

End of Demonstration



Exercises

1. Practicing with Basic Operations

In this exercise, you perform operations on the data used in the previous demonstration. Use the code below at the beginning of your exercise program.

```
proc iml;
  items      ={'Groceries','Utilities','Rent','Car Expenses',
                'Fun Money','Personal Expenses'};
  weeks      ={'Week 1','Week 2','Week 3','Week 4'};
  amounts    ={96 78 82 93,
              61 77 62 68,
              300 300 300 300,
              25 27 98 18,
              55 34 16 53,
              110 85 96 118};
  weeklyIncome = {900 850 1050 950};
  weeklyExpenses = amounts[+,];

```

- a. Create a 1 x 4 matrix named **proportionIncomeSpent** whose elements are the proportion of each week's income that went to expenses. Use the RESET statement to suppress the automatic printing of matrix names. Print the **proportionIncomeSpent** matrix with the values of **weeks** used as column labels and PERCENT7.2 used as a format.

Week 1	Week 2	Week 3	Week 4
71.9%	70.7%	62.3%	68.4%

- b. Create a 1 x 4 matrix named **proportionIncomeSaved** whose elements are equal to the proportion of each week's income that did not go to expenses. That is, use an implicit loop to subtract the values of **proportionIncomeSpent** from 1. Print the **proportionIncomeSaved** matrix with the values of **weeks** used as column labels and PERCENT7.2 used as a format.

Week 1	Week 2	Week 3	Week 4
28.1%	29.3%	37.7%	31.6%

- c. Create a 6 x 4 matrix named **proportionSpentPerItem** whose elements are the proportion of each week's income spent on each item, by week. That is, use an implicit loop to divide the **amounts** matrix by the **weeklyIncome** matrix. Print the **proportionSpentPerItem** matrix with the values of **items** used as row labels, the values of **weeks** used as column labels, and PERCENT7.2 used as a format.

	Week 1	Week 2	Week 3	Week 4
Groceries	10.7%	9.18%	7.81%	9.79%
Utilities	6.78%	9.06%	5.90%	7.16%
Rent	33.3%	35.3%	28.6%	31.6%
Car Expenses	2.78%	3.18%	9.33%	1.89%
Fun Money	6.11%	4.00%	1.52%	5.58%
Personal Expenses	12.2%	10.0%	9.14%	12.4%

- d. Create a matrix named **weeklyExpenseChange** with the same number of rows as **amounts** but with one less column than **amounts** (in other words, a 6×3 matrix). Fill it with missing numeric values. This should be done with a matrix literal. Fill each column of **weeklyExpenseChange** with each column of **amounts** minus the previous column of **amounts**. That is, column 1 of **weeklyExpenseChange** should equal column 2 of **amounts** minus column 1 of **amounts**, and so on. Print a title and print the matrix. Use columns 2 through 4 of **weeks** as column labels and use **items** as row labels.

	Week 2	Week 3	Week 4
Groceries	-18	4	11
Utilities	16	-15	6
Rent	0	0	0
Car Expenses	2	71	-80
Fun Money	-21	-18	37
Personal Expenses	-25	11	22

End of Exercises

7.2 Modules and Subroutines

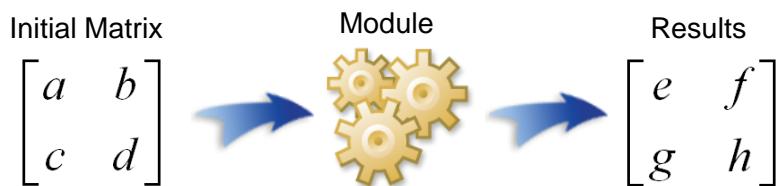
Objectives

- Generate random numbers using both the RANDGEN and RANDFUN modules.
- Identify the differences between a SAS function and subroutine.
- Use built-in functions and subroutines.
- Create user-defined functions and subroutines.
- Store and load modules and matrices.
- Free matrices to save memory.

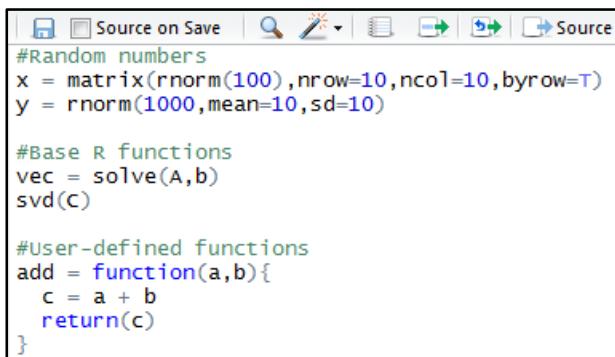
24

Motivation

Use SAS functions and user-defined functions to assist in your IML analyses.



Duplicating the R Script



```

#Random numbers
x = matrix(rnorm(100), nrow=10, ncol=10, byrow=T)
y = rnorm(1000, mean=10, sd=10)

#Base R functions
vec = solve(A,b)
svd(C)

#user-defined functions
add = function(a,b){
  c = a + b
  return(c)
}

```

26

Modules

A module performs a specific task and can be either a function or a subroutine.

General form of an IML function:

```
result = function-name(argument-1, argument-2, ...);
```

- IML functions are not valid without an assignment statement.

General form of an IML subroutine:

```
CALL subroutine-name<(argument-1, argument-2, ...)>;
```

27

IML functions and subroutines perform common operations that would otherwise require many lines of IML code.

Functions versus Subroutines

Function Modules

- return a single matrix
- must have at least one argument
- require an assignment statement

Subroutine Modules

- do not return a matrix
- can have no arguments
- cannot be called in an assignment statement

```
x = {3 4 5,
      2 4 9};
numberRows = nrow(X);
numberCols = ncols(X);
call sort(X);
```

28

Functions versus Subroutines

Some subroutines create matrices without an assignment statement.

General form of the EIGEN subroutine:

```
CALL EIGEN(eigenvalues,eigenvectors,matrix);
```

Example:

```
x = {1 2,3 4};
call eigen(evals,evecs,x);
print evals evecs;
```

Output:

evals	evecs
5.3722813 0 -0.415974 -0.824565	
-0.372281 0 -0.909377 0.5657675	

29

7.03 Poll

The PROC IML code below prints the 0.75 quantile from matrix **X**.

- True
- False

```
Q = call qntl(X,{0.75});
print Q;
```

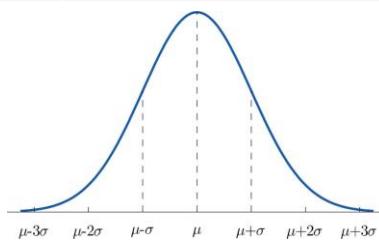
30

Probability Functions

Compare the PDF, CDF, and QUANTILE functions with the R counterparts.

- Use the normal distribution as an example.

R	SAS
dnorm(q)	PDF('Normal',q,mean,sd)
pnorm(q)	CDF('Normal',q,mean,sd)
qnorm(p)	QUANTILE('Normal',p,mean,sd)



32

J Function

The J function creates a matrix with n rows and p columns, with all elements equal to the same value.

$$\boxed{x = J(2, 3, 0);}$$

$$\boxed{J(nrows, ncols, <value>)}$$

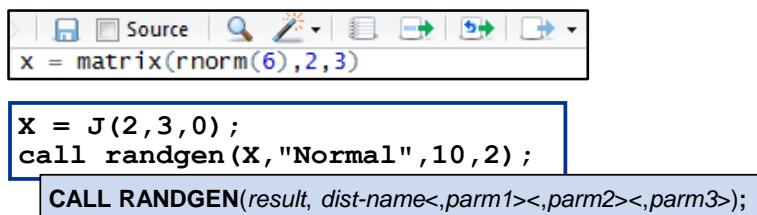

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Value can be character or numeric and is set to 1 by default.

33

CALL RANDGEN

Fill your matrix with random numbers using the RANDGEN subroutine.



```
x = matrix(rnorm(6), 2, 3)
x = J(2, 3, 0);
call randgen(x, "Normal", 10, 2);
CALL RANDGEN(result, dist-name<,parm1><,parm2><,parm3>);
```

- result* is the matrix to be filled with random samples.
- dist-name* is the name of the distribution.

$$X = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \rightarrow \quad X = \begin{bmatrix} 6.09 & 10.61 & 8.58 \\ 6.72 & 7.48 & 12.71 \end{bmatrix}$$

34

The *result* matrix must be created by the user before calling RANDGEN. RANDGEN produces the number of samples required to fill each cell in the matrix. The *result* matrix must be numeric and should have a number of cells equal to the desired number of samples. The number of parameters that are specified is dependent on the distribution. For example, specifying the Cauchy distribution does not require any parameters whereas specifying a normal mixture distribution requires three parameters.

RANDGEN Distributions

The RANDGEN subroutine can be used to generate random numbers from the following built-in univariate distributions:

Bernoulli	Beta	Binomial
Cauchy	Chi-Square	Erlang
Exponential	F	Gamma
Geometric	Hypergeometric	Laplace
Logistic	Lognormal	Negative Binomial
Normal	Normal Mixture	Pareto
Poisson	T	Table
Triangle	Tweedie	Uniform
Wald	Weibull	

35

RANDFUN Function

To simulate random vectors, use the RANDFUN function.

```
Source on Save | Source
x = rnorm(10,10,2)

x = randfun(10,"Normal",10,2);

result = RANDFUN(n, dist-name<,parm1><,parm2><,parm3>);
```

- n is the number of random samples.
- RANDFUN creates an n -by-1 vector.

36

Inside loops, it is more efficient to use the RANDGEN subroutine.



Simple Linear Regression

SP4R07d02.sas

- To practice random number generation and using basic SAS modules and subroutines, create a data matrix corresponding to a simple linear regression and estimate the parameters.

```
proc iml;
  call randseed(27606);
  n = 20;
  beta0 = 5;
  beta1 = 2;

  xvals = j(n,1,0);
  call randgen(xvals,"Uniform");
  xvals = xvals*20;

  error = j(n,1,0);
  call randgen(error,"Normal",0,5);

  y = beta0 + beta1*xvals + error;
  print y beta0 beta1 xvals error;
```

The IML statements above generate 20 values from a simple linear regression model, $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, where $\beta_0 = 5$, $\beta_1 = 2$, $\varepsilon_i \sim N(0, \sigma^2 = 5)$, and the X values are uniformly generated between 0 and 20. Notice that the random numbers were generated using the RANDGEN subroutine to fill the matrices created by the J function. Finally, an implicit loop is used to create the Y vector because **beta0** and **beta1** are scalar values.

y	beta0	beta1	xvals	error
22.256645	5	2	6.9833149	3.2900157
24.970059		11.619663	-3.269267	
37.152607		14.863959	2.424689	
43.883992		17.563605	3.7567819	
3.863763		0.0771578	-1.290553	
33.306874		16.60792	-4.908965	
29.346249		14.251058	-4.155868	
50.956811		14.773279	16.410253	
5.1425732		1.6650986	-3.187624	
18.32976		11.628301	-9.926843	
48.614374		14.12347	15.367434	
12.880698		1.5470275	4.7866428	
42.572556		19.547	-1.521443	
31.325892		12.918223	0.4894454	
1.9870184		1.3057261	-5.624434	
29.396601		8.558932	7.2787367	
18.379377		6.7389815	-0.098586	
9.877005		0.425611	4.025783	
3.4572365		0.1731237	-1.889011	
28.905527		13.190648	-2.475768	

2. Create the design matrix and the $X^T X$ matrix.

```
x = j(n,1,1)||xvals;
xpx = x`*x;
print x, xpx;
```

The design matrix is created by creating a vector of 1s with the J function and concatenating it with the X values. The $X^T X$ matrix is created using the transpose operator, ` , and the SAS matrix multiplication operator, * .

x

```
1 6.9833149
1 11.619663
1 14.863959
1 17.563605
1 0.0771578
1 16.60792
1 14.251058
1 14.773279
1 1.6650986
1 11.628301
1 14.12347
1 1.5470275
1 19.547
1 12.918223
1 1.3057261
1 8.558932
1 6.7389815
1 0.425611
1 0.1731237
1 13.190648
```

xpx

```
20 188.5621
188.5621 2593.7719
```

3. Use the SVD subroutine to create a singular value decomposition and reassemble the $X^T X$ matrix.

```
call svd(u,q,v,xpx);
xpxSVD = u*diag(q)*v`;
print u q v xpxSVD;
```

The SVD subroutine outputs the matrices \mathbf{U} , \mathbf{Q} , and \mathbf{V} . Here, \mathbf{Q} is a vector of singular values. Recall that the singular value decomposition is formed back into the original matrix by the operations UDV^T , where D represents a diagonal matrix of \mathbf{Q} .

u	q	v	xpxSVD
0.0726811 0.9973552 2607.5131 0.0726811 0.9973552			20 188.5621
0.9973552 -0.0726811 6.2587499 0.9973552 -0.0726811			188.5621 2593.7719

4. Create the inverse of the \mathbf{Q} matrix.

```
qInv = 1/q;
qInvDiag = diag(qInv);
print q qInv qInvDiag;
```

Recall that the inverse of a matrix and the singular value decomposition representation is $Z^{-1} = (UDV^T)^{-1} = VD^{-1}U^T$. The vector **qInv** is created using an implicit operator and the **DIAG** function is used to create diagonal elements of the **qInv** vector.

q	qInv	qInvDiag
2607.5131	0.0003835	0.0003835
6.2587499	0.1597763	0.1597763

5. Calculate parameter estimates using the singular values decomposition to invert the $X^T X$ matrix.

```
betaHat = (v*qInvDiag*u`)*(x`*y);
print betaHat;
quit;
```

Compute $\hat{\beta} = (X^T X)^{-1} X^T Y$ and print the results.

betaHat
4.7235927
2.1326331

End of Demonstration

Motivation

What are some other useful modules?

Mathematical	ABS, EXP, LOG, SQRT
Reduction	MAX, MIN, PROD, SUM
Matrix Inquiry	ALL, ANY, LOC, COUNTN
Matrix Reshaping	VECDIAG, REPEAT, SHAPE
Random Number Generation	CALL RANDGEN, CALL RANDSEED
Statistical	CORR, COV, MEAN, CALL QNTL
Numerical Analysis	CALL SPLINE, BSPLINE
Linear Algebra	DET, TRACE, INV, CALL EIGEN, SOLVE, CALL SVD, CALL QR, ROOT
Optimization	CALL NLPNRA
...	...

38

Most modules are intuitive and identical to R. For example, the ABS() function returns the absolute value for each element in a matrix and is identical to R syntax.

Matrix Reshaping

Use the REPEAT function to duplicate **rep()** in R.

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
Y = repeat(X, 2, 2);
```

```
result = REPEAT(matrix,nrow,ncol);
```

$$Y = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{bmatrix}$$

```
Y = repeat(X, {2 2 3 3});
```

```
result = REPEAT(matrix,freq);
```

$$Y = [1 \ 1 \ 2 \ 2 \ 3 \ 3 \ 3 \ 4 \ 4 \ 4]$$

39

The first form of the REPEAT function creates a new matrix by repeating the values of matrix *nrow* times across the rows and *ncol* times across the columns.

The second form of the REPEAT function returns a row vector with each value in *matrix* repeated the number of times specified in *freq*.

CONCAT Function

General form of the CONCAT function:

```
result = CONCAT(matrix1, matrix2,...);
```

- The CONCAT function returns a character matrix.
- Only character matrices can be used as arguments.

```
pre = j(1,3,"data");
post = char(1:3);
result = concat(pre,post);
```

$$\text{result} = [\text{data 1} \quad \text{data 2} \quad \text{data 3}]$$

40

The CHAR() function returns a character representation of a numeric matrix.

Matrix Inquiry Functions

The ANY(), ALL(), and ISEMPTY() functions return a value of 1 or 0 according to specified criteria.

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

A = any(X>3); **A = [1]**

B = all(X>3); **B = [0]**

C = isempty(Y); **C = [0]**

41

SAMPLE and UNIQUE Functions

The SAMPLE and UNIQUE functions are identical to R.

$$X = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

```
p = c(.75,.2,.05)
y = sample(x,10,replace=TRUE,prob=p)
```

```
p = { .75,.2,.05};
y = sample(x,10,"Replace",p);
```

```
result = SAMPLE(matrix, n, <method>,<prob>);
```

```
y = unique(x)
```

```
y = unique(x);
```

```
result = UNIQUE(matrix);
```

42

The SAMPLE function generates a random sample of the elements of the matrix. The SAMPLE function *method* can be “REPLACE”, “NOREPLACE”, or “WOR”. The *method* of “WOR” specifies a simple random sampling without replacement. After the elements are randomly selected, their order is randomly permuted. The *prob* argument is a vector with the same number of elements as the matrix. The vector specifies the sampling probability for the elements of the matrix. The SAMPLE function scales the elements of *prob* so that they do not need to sum to 1.

The UNIQUE function returns a row vector with the sorted set of all elements in the matrix without duplicates. The matrix can be either numeric or character.

The COUNTUNIQUE function returns the number of unique values in a matrix, or the length of the returned matrix from the UNIQUE function.

Linear Algebra Modules

General form of linear algebra modules:

INV(X)

» Computes the inverse of a square nonsingular matrix.

SOLVE(A,B)

» Solves a system of linear equations.

ROOT(X)

» Performs the Cholesky decomposition of a symmetric positive definite matrix.

GINV(X)

» Computes the Moore-Penrose generalized inverse of a matrix.

SVD(X)

» Computes the singular value decomposition of a matrix.

43

Moral of the Story...

When in doubt, search the web!

- The SAS documentation lists all IML functions and subroutines.
- Visit the SAS/IML blog to learn tips and tricks with SAS/IML: <http://blogs.sas.com/content/iml/>
- *Statistical Programming with SAS/IML® Software* provides a comprehensive description of the software and how to use it.
 - Author: Rick Wicklin

44



Navigating the SAS/IML Documentation

1. Go to support.sas.com/documentation.
2. Select **SAS Analytical Products 14.1**.
3. Go to the **SAS/IML 14.1** category and select **What's New in SAS/IML 14.1**.
4. On the Contents tab, scroll to the **Language Reference** category and select the gray box to expand the category.

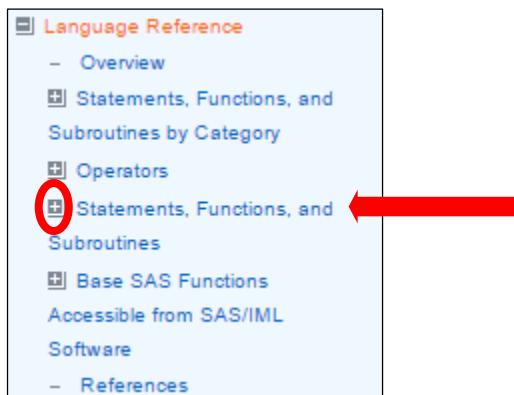
SAS/IML(R) 14.1 User's

PDF

Contents [About](#)

- [+ Programming Statements](#)
- [+ Working with SAS Data Sets](#)
- [+ File Access](#)
- [+ Packages](#)
- [+ General Statistics Examples](#)
- [+ Submitting SAS Statements](#)
- [+ Calling Functions in the R Language](#)
- [+ Robust Regression Examples](#)
- [+ Time Series Analysis and Examples](#)
- [+ Nonlinear Optimization Examples](#)
- [+ Statistical Graphics](#)
- [+ Traditional Graphics in the IML Procedure](#)
- [+ Window and Display Features](#)
- [+ Storage Features](#)
- [+ Using SAS/IML Software to Generate SAS/IML Statements](#)
- [+ Wavelet Analysis](#)
- [+ Genetic Algorithms](#)
- [+ Sparse Matrix Algorithms](#)
- [+ Further Notes](#)
- [+ Language Reference](#) 
- [+ Module Library](#)

5. Select the gray box next to **Statements, Functions, and Subroutines** to expand it.



This lists all the statements, functions, and subroutines that are available in SAS/IML.

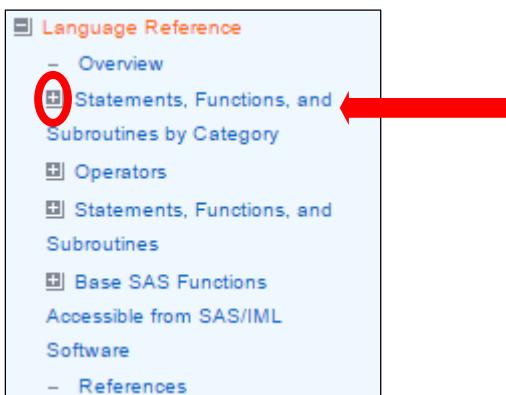
6. Select **NLPNRA Call**.

Several SAS/IML optimization subroutines are available to the user. Each one of these subroutines begins with NL, for nonlinear optimization. The documentation provides details about the subroutines as well as an example that users can try in their own SAS/IML session.

```
start F_BETTS(x);
  f = .01 * x[1] * x[1] + x[2] * x[2] - 100;
  return(f);
finish F_BETTS;

con = { 2 -50 . ..,
        50 50 . ..,
        10 -1 1 10};
x = {-1 -1};
opt = {0 2};
call nlpnra(rc, xres, "F_BETTS", x, opt, con);
```

7. In the Language Reference area, select the gray box next to **Statements, Functions, and Subroutines by Category**.



You can easily find SAS/IML tools with the following categories:

The screenshot shows a vertical list of menu items under the heading "Language Reference". The items are organized into several main categories, each with a small icon to its left. The categories are:

- Overview
- Statements, Functions, and Subroutines by Category
 - Mathematical Functions
 - Reduction Functions
 - Matrix Inquiry Functions
 - Matrix Sorting and BY-Group Processing Functions
 - Matrix Reshaping Functions
 - Combinatorial Functions
 - Character Manipulation Functions
 - Functions for Generating Random Numbers and Simulations
 - Statistical Functions
 - Time Series Functions
 - Numerical Analysis Functions
 - Linear Algebra functions
 - Optimization Subroutines
 - Set functions
 - Control Statements
 - Data Set and File Statements
 - Defining, Storing, and Loading Modules
 - Packages and Executing Statements
 - Statistical Graphics
 - Termination Statements
 - Traditional Graphics and Window functions
 - Wavelet Analysis functions
 - Genetic Algorithm functions
 - Calling External Modules
 - Calling SAS statements or R Functions

End of Demonstration

7.04 Quiz

Navigate to the SAS/IML documentation and peruse the statements, functions, and subroutines. Choose a few that look familiar to you and see what they do. Next, find the LOC function and see what it does.

46

Create a Module

Create a module to do the following tasks:

- implement analyses from a textbook and from journal articles
- implement proprietary algorithms that were developed at your company
- assist in creating any type of IML script

48

Idea Exchange

What type of R functions do you make when scripting?



49

Defining a Module

Compare the syntax between R and SAS.

```
Source on Save | Source
function.name = function(arg1,arg2,...){
  statements
  return(arguments)
}
```

```
START name <(argument1, argument2,...)>
  <GLOBAL(argument1, argument2,...)>;
  statements;
  <RETURN(matrix);>
FINISH;
```

- *name* is the user-defined module name.
- *arguments* are input or output matrices (or both) that are used or created by the module.

50

A module always begins with the START statement and ends with the FINISH statement. The START statement instructs IML to enter a module-collect mode. In this mode, IML gathers the statements of a module rather than executing them immediately. The FINISH statement signals the end of a module.

Function Modules

Function modules

- return a single matrix
- require the RETURN statement
- are executed using an assignment statement.

```
start add(a,b);
  c = a + b;
  return(c);
finish;

x = {1 2, 3 4};
y = {5 6, 7 8};
z = add(x,y);
```

51

Function Modules

The GLOBAL clause is used to specify variables that are used in the module but not specified as inputs.

The screenshot shows an RStudio interface with the following code:

```
x = matrix(c(1,2,3,4),nrow=2,ncol=2,byrow=T)
y = matrix(c(5,6,7,8),nrow=2,ncol=2,byrow=T)

add = function(a){
  c = a + y
  return(c)
}

x = {1 2, 3 4};
y = {5 6, 7 8};

start add(a) global(y);
  c = a + y;
  return(c);
finish;

z = add(x);
```

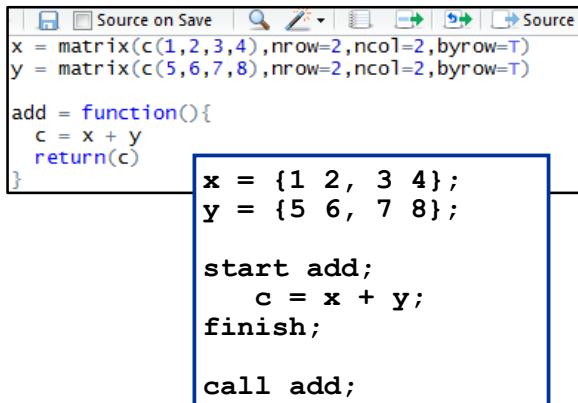
The code defines two matrices `x` and `y`, a function `add` that adds `x` and `y`, and then uses the `add` function to calculate `z`. The `add` function includes a `global(y)` clause to make `y` available within the function's scope.

52

R automatically passes variable names into functions.

Subroutine Modules

Ignore the GLOBAL clause if there are no inputs.



```

x = matrix(c(1,2,3,4),nrow=2,ncol=2,byrow=T)
y = matrix(c(5,6,7,8),nrow=2,ncol=2,byrow=T)

add = function(){
  c = x + y
  return(c)
}

x = {1 2, 3 4};
y = {5 6, 7 8};

start add;
  c = x + y;
finish;

call add;

```

53

Subroutine Modules

Subroutine modules

- do not return a matrix
- do not use a RETURN statement
- are not used in an assignment statement
- are executed using a CALL (or RUN) statement.

Recall the EIGEN subroutine.

```

x = {1 2,3 4};
call eigen(evals,evecs,x);
print evals evecs;

```

54

Subroutine Modules

Proper subroutine module syntax:

- First provide the output matrices.
- Then provide the input matrices.

```
start addsub(x,y,a,b);
  x = a + b;
  y = a - b; ← (local symbols)
finish;

matOne = {1 2, 3 4};
matTwo = {5 6, 7 8};

call addsub(add,sub,matOne,matTwo);
```

55

$$add = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix} \quad sub = \begin{bmatrix} -4 & -4 \\ -4 & -4 \end{bmatrix}$$

Subroutine modules are used to output multiple matrices. This is similar to returning a list in R.

The symbols X, Y, A, and B are local symbols, meaning that they are not recognized outside of the user-defined module. As a result, you can specify any symbol as the output matrices in the CALL statement.

7.05 Multiple Answer Poll

Which of the following statements about SAS modules are true?

- a. Modules are defined by START and FINISH keywords.
- b. Functions use the RETURN statement
- c. The RETURN statement can handle multiple arguments.
- d. Subroutines can be executed by the CALL statement.

56

Library Storage

Library storage can be used to store user-defined modules and the values of matrices on disk for later retrieval.

- Store and reload IML modules and matrices.
- Save work for a later session.
- Conserve memory by saving large, intermediate results for later use.

58

Storage Catalogs

default library: **work**

default storage catalog: **imlstor**

RESET STORAGE = </libref.>catalog;
 specifies the library and catalog names.
SHOW STORAGE;
 lists all entries currently in storage.

59

SAS/IML storage catalogs are specially structured SAS files that are located in a SAS library. A SAS/IML catalog contains *entries* that are either matrices or modules. Like other SAS files, SAS/IML catalogs have two-level names in the form *libref.catalog*. The first-level name, *libref*, is a name assigned to the SAS library to which the catalog belongs. The second-level name, *catalog*, is the name of the catalog file.

When you store a matrix, IML automatically stores the matrix name, its type, its dimension, and its current values.

Modules are stored in the form of their compiled code. After modules are loaded, they do not need to be parsed again, making their use very efficient.

The default libref is initially **work**, and the default catalog is **imlstor**. Thus, the default storage catalog is called **work.imlstor**. You can change the storage catalog (or both the library reference and catalog) with the RESET STORAGE command. You can list all modules and matrices in the current storage catalog using the SHOW STORAGE command.

Managing an IML Catalog

General forms of catalog management statements:

```
keyword matrices;  
keyword MODULE=(modules);  
keyword MODULE=(modules) matrices;
```

keyword can be any of the following:

- LOAD recalls entries back into the IML workspace.
- REMOVE deletes entries from the catalog.
- STORE places IML modules, matrices, or both into catalog storage.

60

These statements enable you to manage your catalog entries:

LOAD	recalls entries from storage into IML.
REMOVE	removes entries from storage.
STORE	saves modules or matrices to storage.
RESET STORAGE= <i>libref.catalog</i>	specifies the two-level catalog name.
SHOW STORAGE	lists all current entries.

Notice the following:

- A statement with no operands acts on all modules and matrices.
- The special operand **_ALL_** can be used to specify all modules, all matrices, or all modules **and** matrices.
- If only one module is specified, then the parentheses around the module name are not required.

STORE Statement

```
reset storage=sp4r.cat1;  
  
store expense;  
  
store module=impute;  
  
store module=_all_;  
  
store module=(impute) x y;  
  
store;
```

-  The keyword STORE can be replaced with either LOAD or REMOVE, and the syntax still holds.

61

The RESET statement specifies the storage catalog to be in libref **sp4r** with the catalog name **cat1**.

The STORE EXPENSE statement saves the matrix **EXPENSE** onto disk, in **sp4r.cat1**.

The STORE MODULE=IMPUTE statement saves the user-defined module IMPUTE in **sp4r.cat1**.

The STORE MODULE=_ALL_ statement saves all modules in the current IML session in **sp4r.cat1**.

The STORE MODULE=(IMPUTE) X Y statement saves the module IMPUTE and the matrices **x** and **y** in **sp4r.cat1**.

The STORE; statement saves all matrices and modules in the current IML session in **sp4r.cat1**. This can help you save your complete IML environment before exiting an IML session. Then you can use the LOAD statement in a subsequent session to restore the environment and resume your work.

The LOAD command can be used to restore matrices and modules from storage back into the IML active workspace.

The REMOVE command can be used to remove modules or matrices that are no longer needed from the catalog. The REMOVE command has the same form as the STORE command and the LOAD command.

7.06 Multiple Choice Poll

How do you recall the module **rock** and the matrix **pony** into a new SAS/IML session from the **mycat** catalog in the **Work** directory?

- a. RESET STORAGE; LOAD module=(rock) pony;
- b. RESET STORAGE=mycat; LOAD module=(rock) pony;
- c. STORAGE=mycat; LOAD module=(rock) pony;
- d. RESET STORAGE=mycat; LOAD rock pony;

62

Not Enough Memory?

IML is an in-memory procedure. All IML matrices are stored in RAM.

- Use the FREE statement to free matrices that are no longer needed.
- Use the STORE statement to store matrices on disk and then use the FREE statement to free their values. Restore the matrices later using the LOAD statement.
- Reformulate your approach to use smaller matrices (for example, by using VAR and WHERE clauses where applicable).

64

The amount of memory that your system can provide depends on the capacity of your computer and on the other products that are installed. IML does everything in memory, using both virtual memory and RAM. Each matrix requires (#rows * #columns * 8) + 16 bytes for storage.



The bulleted items on the slide above are rarely needed for modern computers that have 8GB or more of RAM.

FREE Statement

General form of the FREE statement:

```
FREE matrices;  
FREE / <matrices>;
```

FREE X Y; → Free the matrices **X** and **Y**.

FREE / ; → Free all matrices.

FREE / A B; → Free all matrices except for
A and **B**.

65

The FREE statement frees matrix storage spaces to make room for more data (matrices) in the workspace. The FREE statement is used mostly in large applications or under tight memory constraints.

Notice that the STORE statement stores only matrices and modules. It does not free them from memory, so you can still reference them later in the same IML session. If you also issued the FREE statement afterward, the matrices are no longer in the memory, and you must use the LOAD statement to restore them.



Creating Functions and Subroutines

SP4R07d03.sas

1. Extend the simple linear regression demonstration by creating functions and subroutines for the process. Start by creating a function to return parameter estimates of a simple linear regression model with input X and Y values.

```
proc iml;
  start simpleRegFunc(xvals, yvals);
    n = nrow(xvals);
    x = j(n,1,1)||xvals;
    y = yvals;

    betaHat = inv(x`*x)*(x`*y);
    return(betaHat);
  finish;
```

The SimpleReg function takes in both the independent and dependent values as inputs and returns the parameter estimates. Notice that only one matrix is supplied to the RETURN statement.

2. Generate some data to supply the new function.

```
call randseed(27606);
n = 20;
beta0 = 5;
beta1 = 2;
xvals = randfun(n,"Uniform");
xvals = xvals*20;
error = randfun(n,"Normal",0,5);
y = beta0 + beta1*xvals + error;

betas = simpleRegFunc(xvals,y);
print betas;
```

The IML statements above generate 20 values from a simple linear regression model, $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, where $\beta_0 = 5$, $\beta_1 = 2$, $\varepsilon_i \sim N(0, \sigma^2 = 5)$, and the X values are uniformly generated between 0 and 20. The generated X values and the response are then passed to the user-defined function and the output matrix is assigned to the matrix betas.

betas
4.7235927
2.1326331

4.7235927
2.1326331

3. Alter the SimpleReg function by creating a subroutine that outputs both the parameter estimates of $\hat{\beta}$ as well as $\hat{\sigma}$.

```
start simpleRegSub(betaHat, sigmaHat, xvals, yvals);
  n = nrow(xvals);
  x = j(n,1,1)||xvals;
  y = yvals;

  betaHat = inv(x`*x)*(x`*y);
  pred = x*betaHat;
  sse = sum( (y-pred)##2 );
  sigma2Hat = sse / (n-1);
  sigmaHat = sqrt(sigma2Hat);
finish;
```

Notice that BetaHat and SigmaHat are specified before the inputs in the user-defined subroutine. This is proper syntax for a SAS subroutine. The subroutine requires a CALL statement to run the module. The output matrices can be named anything that the user specifies.

4. Use the subroutine on the generated data and print the results.

```
call simpleRegSub(betas,sHat,xvals,y);
print betas sHat;
```

betas	sHat
4.7235927	6.4943754
2.1326331	

5. Use the RESET and STORE statements to save all the matrices and modules from the IML session in a catalog named **imlcat** in the **Work** library.

```
reset storage=imlcat;
store;
show storage;
quit;
```

Contents of storage library = WORK.IMLCAT

Matrices:	
BETA0	BETA1
BETAS	ERROR
N	SHAT
XVALS	Y
Modules:	
RANDFUN	SIMPLEREGFUNC
SIMPLEREGSUB	

End of Demonstration

7.3 Calling SAS Data Sets and Procedures

Objectives

- Import a SAS data set into an IML matrix using the USE, READ, and CLOSE statements.
- Export an IML matrix to a SAS data set using the CREATE, APPEND, and CLOSE statements.
- Add an IML matrix to an existing SAS data set with the EDIT statement.
- Use the SUBMIT statement to call SAS procedures from the IML environment.

68

Motivation

Use IML to perform the following tasks:

- create a matrix using a SAS data set



- export a matrix to a SAS data set



- add data to an existing SAS data set



69

In R, the data frame is already a matrix and ready for matrix computation.

What Is an Open Data Set?

An *open* data set is one that is ready for Read or Write access (or both). You can use one of three statements to open a data set:

- USE enables Read access to an existing data set.
- EDIT enables Read and Write access to an existing data set.
- CREATE creates a new data set for both Read and Write access.

To *close* an open data set, simply use the CLOSE statement.

- CLOSE closes an open data set.

70

If you forget to use the CLOSE statement, SAS closes the open data set when you exit SAS/IML with the QUIT statement.

If you open a data set with the USE statement, you can still open the data set manually to view the table. However, opening a data set with the EDIT statement does not permit you to open and view the data table manually.

USE, READ, and CLOSE

To create a matrix using a SAS data set use the statements USE, READ, and CLOSE.



```
use class;
read all var {height weight}
  where (sex='M') into imlClass;
close class;
```

71

READ Statement

```
use class;
read all var {height weight}
  where (sex='M') into imlClass;
close class;

READ <range> <VAR operand> <WHERE(expression)>
  <INTO name <[ROWNAME=row-name COLNAME=col-name]>>;
```

- *range* specifies a range of observations:
ALL CURRENT (default)
NEXT *n* AFTER POINT *observation(s)*
- *operand* selects a set of variables.
- *expression* is evaluated as true or false.
- *name* is a target matrix to be created whose values are read from the current data set.

72

The VAR option takes a keyword (for example, _ALL_) or a vector of variable names as arguments. The VAR option cannot take column numbers as arguments.

READ Statement

```
use class;
read all var {height weight}
  where (sex='M') into imlClass;
close class;

READ <range> <VAR operand> <WHERE(expression)>
  <INTO name <[ROWNAME=row-name COLNAME=col-name]>>;
```

- *row-name* is the name of a character variable in the input data set giving descriptive row labels. A new matrix (column vector) *row-name* is created as well.
- *col-name* is a new character matrix (row vector) to be created containing variable names from the input data set corresponding to columns of the target matrix.

73

CREATE and EDIT Statements

To create a new SAS data set from an IML matrix, use the CREATE and EDIT statements.

- The CREATE statement creates a new data set.



- The EDIT statement enables the user to add data to an existing data set.



74

Creating a Data Set with the CREATE Statement

General form of the CREATE statement:

First Form

```
CREATE SAS-data-set <VAR {operand}>;
```

operand gives a set of existing IML matrices to become data set variables.

75

If the VAR option is not used, a variable is created for every SAS/IML matrix that is in scope, and the matrix names are used as variable names in the new data set.

Each matrix used to create the data set corresponds to a single variable in the data set. If a matrix with multiple rows and multiple columns is used as a data set variable, its contents are written to the data set in row-major order.

Creating a Data Set with the CREATE and FROM Statements

General form of the CREATE statement:

Second Form

```
CREATE SAS-data-set FROM matrix-name  

  <[ROWNAME=row-name  

   COLNAME=column-name]>;
```

- | | |
|---------------------------------------|--|
| <i>row-name</i>
<i>column-name</i> | an existing character matrix or quoted literal that adds a variable to the data set to contain row titles
a character matrix or quoted literal providing the variable names in the data set being created |
|---------------------------------------|--|

76

When the FROM keyword is used in the CREATE statement, each column in the source matrix is treated as a distinct variable in the newly created data set.

Adding Observations with APPEND

The CREATE statement opens a data set only for input or output. You need to use the APPEND statement to write to the data set.

General forms of the APPEND statement:

```
APPEND;
```

```
APPEND VAR matrix-list;
```

```
APPEND FROM append_matrix <[ROWNAME=row-name]>;
```

77

- | | |
|--|---|
| APPEND
APPEND VAR matrix-list | adds data from matrices already associated with the data set.
This cannot be combined with CREATE FROM statement.
adds data from a subset of matrices already associated with the data set. This cannot be combined with CREATE FROM statement. |
|--|---|

APPEND FROM *append_matrix* adds data from a specified matrix. The matrix whose data are appended does not need to be the same matrix that was used to create the data set, but must be of the same type (character or numeric) and must have the same number of columns.

CREATE, APPEND, AND CLOSE Statements

To create a SAS data set using an IML matrix, use the statements CREATE, APPEND, and CLOSE.

```
create data1 var{name age};  
append;  
close data1;
```

```
create data2;  
append var{name age};  
close data2;
```

```
create data3 from matrix3  
[colname={week1, week2, week3, week4}];  
append from matrix3;  
close data3;
```

78

The VAR option can be used in either the CREATE statement or the APPEND statement but not both. The VAR option specifies which vectors to pass to a SAS data set. The VAR option is not used to pass a matrix to a SAS data set.

The FROM option is used in both the CREATE and APPEND statements when the user is creating a SAS data set from an IML matrix. For this method, the COLNAME= option must be used in the CREATE statement to identify the names of the SAS data set variables. If it is not used, the variable names in the SAS data set default to COL1, COL2, and so on.

7.07 Multiple Answer Poll

Which statements are true regarding importing SAS data sets and exporting IML matrices? (Select all that apply.)

- a. The statements USE, READ, and CLOSE are used to pass a SAS data set into IML.
- b. The statements CREATE, APPEND, and CLOSE are used to pass an IML matrix to a SAS data set.
- c. Names of IML vectors are passed to the SAS data set as variable names.
- d. The user must specify the column names when creating a SAS data set from an IML matrix.

79

Editing a Data Set

- If a data set already exists, you can open the data set with the EDIT statement, add data, and then close the data set.

```
edit data2;
append from matrix4;
close data2;
```

- The appended matrix must have the same number of columns as the variables in the data set.
 - The matrix is stacked vertically.



81



Calling SAS Data Sets from SAS/IML

SP4R07d04.sas

1. Compute the means of the variables **msrp**, **mpg_city**, and **mpg_highway** for the **cars** data set in SAS/IML for cars greater than \$85,000 in value.

```
proc iml;
  use sp4r.cars;
  read all var {msrp mpg_city mpg_highway}
    where(msrp>80000) into imlCars;
  close sp4r.cars;
```

Each data set pulled into IML should start with the USE statement and end with the CLOSE statement. The WHERE option in the READ statement is used to condition on the variable **msrp**.

```
varNames = {"MSRP" "MPG City" "MPG Highway"};
Means = mean(imlCars);
print imlCars, Means[colname=varNames];
quit;
```

The **varNames** assignment is used for the PRINT statement. Recall that the MEAN function is identical to the **colMeans** function in R. A reduction operator could be used instead.

imlCars		
89765	17	24
84600	15	22
81795	12	20
81995	16	23
86995	16	23
94820	16	24
128420	13	19
86970	16	24
90520	16	23
121770	14	21
126670	13	19
84165	17	24
192465	17	24

Means		
MSRP	MPG City	MPG Highway
103919.23	15.230769	22.307692

2. Practice using the CREATE and APPEND statements to create a new SAS data set.

Create a matrix with 10 rows and three columns in IML and then export it to a SAS data set. Let the first column of random numbers be generated from $N(\mu = 200, \sigma = 50)$, the second column from $Poisson(\lambda = 35)$, and the third column be a random sample of colors.

```
proc iml;
  call randseed(802);
  n=10;
  weight = randfun(n,"Normal",200,50);
  age = randfun(n, "Poisson",35);
```

The variables **weight** and **age** are sampled independently with the RANDFUN function.

```

create sp4r.newDataTable var{age weight};
append;
close sp4r.newDataTable;

/*Identical Result 2
create sp4r.newDataTable2;
append var{age weight};
close sp4r.newDataTable2;*/
quit;

proc print data=sp4r.newDataTable; run;

```

The three vectors are passed to a SAS data set using the VAR option in the CREATE statement. In this case, nothing follows the APPEND statement. An alternative method is shown where the VAR option is removed from the CREATE statement and added to the APPEND statement.

Obs	AGE	WEIGHT
1	33	266.069
2	39	202.331
3	37	200.700
4	35	242.352
5	37	225.553
6	34	217.022
7	47	139.812
8	37	210.402
9	35	179.341
10	34	269.633

The third alternative to create the data set as seen below.

```

/*Identical Result 3
mymat = age||weight;
print mymat;

create newDataTable from mymat [colname={weight, age}];
append from mymat;
close newDataTable;*/

```

Because the vectors are concatenated to make a 10 x 2 matrix first, the CREATE statement needs the COLNAME= option to create the data set. Otherwise, the variable names are not passed to the SAS data set.

3. Increase the size of the data set by sampling more values of **weight** and **age** in IML and adding it to the existing SAS data set.

```

proc iml;
  call randseed(919);
  n=10;
  weight = randfun(n,"Normal",200,50);
  age = randfun(n, "Poisson",35);
  mymat = weight||age;

  edit sp4r.newDataTable;
  append from mymat;
  close sp4r.newDataTable;
quit;

proc print data=sp4r.newDataTable; run;

```

The APPEND statement stacks the existing data set on top of the IML matrix. Thus, the number of columns in the IML matrix must be the same as the number of variables in the SAS data set.

Obs	AGE	WEIGHT
1	33	266.069
2	39	202.331
3	37	200.700
4	35	242.352
5	37	225.553
6	34	217.022
7	47	139.812
8	37	210.402
9	35	179.341
10	34	269.633
11	33	221.042
12	28	226.415
13	34	198.035
14	36	209.033
15	28	252.583
16	31	241.187
17	40	255.169
18	35	145.516
19	33	210.594
20	44	173.695

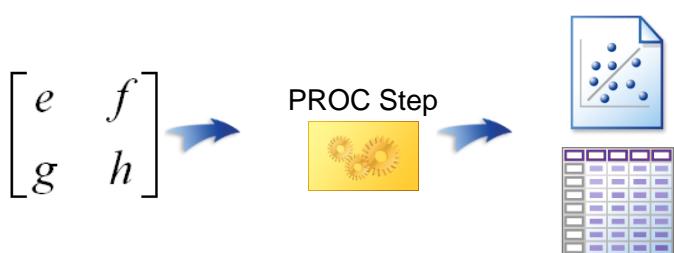


To append new variables to an existing data set, create a new data set and then merge the two data tables.

End of Demonstration

Motivation

- Use SAS procedures on IML matrices.



83

Benefits of Calling SAS Procedures from IML

- You can call SAS procedures without exiting IML.
- SAS procedures can be used within IML modules.
- Matrix values can be used as parameters for SAS procedures.
- You can execute SAS procedures conditionally or within loops.

84

SUBMIT Block

General form of a SUBMIT block:

```
SUBMIT <parameters> / <options>;  
      statements;  
ENDSUBMIT;
```

- The statements between the SUBMIT and ENDSUBMIT statements are referred to as a *SUBMIT block*.
- The *parameters* value specifies one or more option matrices whose values are substituted into language statements in the SUBMIT block.

85

SUBMIT Block Contents

General form of a SUBMIT block:

```
SUBMIT <parameters> / <options>;  
      statements;  
ENDSUBMIT;
```

- SUBMIT blocks can contain the following:
 - SAS procedures
 - DATA steps
 - ODS commands
 - other SAS statements
- A SUBMIT block executes only after the ENDSUBMIT; line is run.

86

Statistical Graphics in SAS/IML

SAS/IML provides subroutines that enable you to create statistical graphics. These subroutines use the SUBMIT and ENDBUSMIT statements to call PROC SGPlot.

- BAR call
- BOX call
- HISTOGRAM call
- SERIES call
- HEATMAPCONT call
- HEATMAPDISC call

87

Go to the following web page to see an overview of statistical graphics in SAS/IML:

http://support.sas.com/documentation/cdl/en/imlug/68150/HTML/default/viewer.htm#imlug_graphics_sect001.htm

In addition, each graphics subroutine provides options to alter the display of the plot. This is similar to the SGPlot procedure.

The heat map subroutines enable the user to visualize data in a matrix.

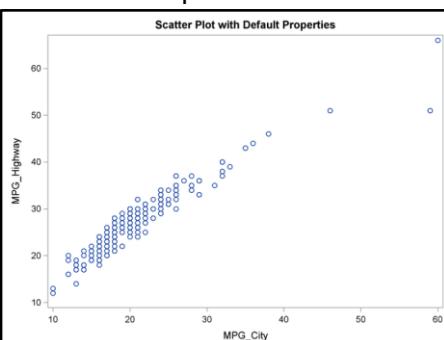
HEATMAPCONT creates a heat map with continuous color map.

HEATMAPDISC creates a heat map with a discrete color map.

SCATTER Subroutine

Form of the SCATTER subroutine and options:

```
CALL SCATTER (x,y)
GROUP=GroupVector
DATALABEL=LabelVector
OPTION=ScatterOption
GRID={"X" <,"Y">}
LABEL={"XLabel" <,"YLabel">}
XVALUES=xValues
YVALUES=yValues
PROCOPT=ProcOption
LINEPARM=(x0 y0 slope)
OTHER=stmts;
```



```
title "Scatter Plot with Default Properties";
call Scatter(MPG_City, MPG_Highway)
label={"MPG_City" "MPG_Highway"};
```

88

Possible SCATTER subroutine options:

GROUP=	specifies a vector of values that determine groups in the plot.
DATALABEL=	specifies a vector of values that label each marker in the plot.
OPTION=	specifies a character matrix of string literal. The value is used verbatim to specify options in the SCATTER statement of the SGLOT procedure.
GRID=	specifies whether to display grid lines for the X or Y axis.
LABEL=	specifies axis labels for the X or Y axis.
XVALUES=	specifies a vector of values for ticks for the X axis.
YVALUES=	specifies a vector of values for ticks for the Y axis.
PROCOPT=	specifies options in the PROC SGLOT statement.
OTHER=	specifies statements in the SGLOT procedure.



Submitting SAS Procedures

SP4R07d05.sas

Use the simple linear regression script to estimate the model parameters and submit a SAS procedure to ensure the results.

1. Use the simple linear regression script from an earlier demonstration.

```
proc iml;
  call randseed(27606);
  n = 20;
  beta0 = 5;
  beta1 = 2;
  xvals = randfun(n,"Uniform");
  xvals = xvals*20;
  error = randfun(n,"Normal",0,5);
  y = beta0 + beta1*xvals + error;
  x = j(n,1,1)||xvals;
  betaHat = inv(x`*x)*(x`*y);
  print betaHat;
```

The IML script is identical to a previous demonstration.

```
betaHat
4.7235927
2.1326331
```

2. Export the dependent and independent variables to a SAS data set.

```
create sp4r.betaData var{xvals y};
append;
close sp4r.betaData;
```

3. Submit PROC REG code in SAS/IML to ensure that the estimates of BetaHat are correct.

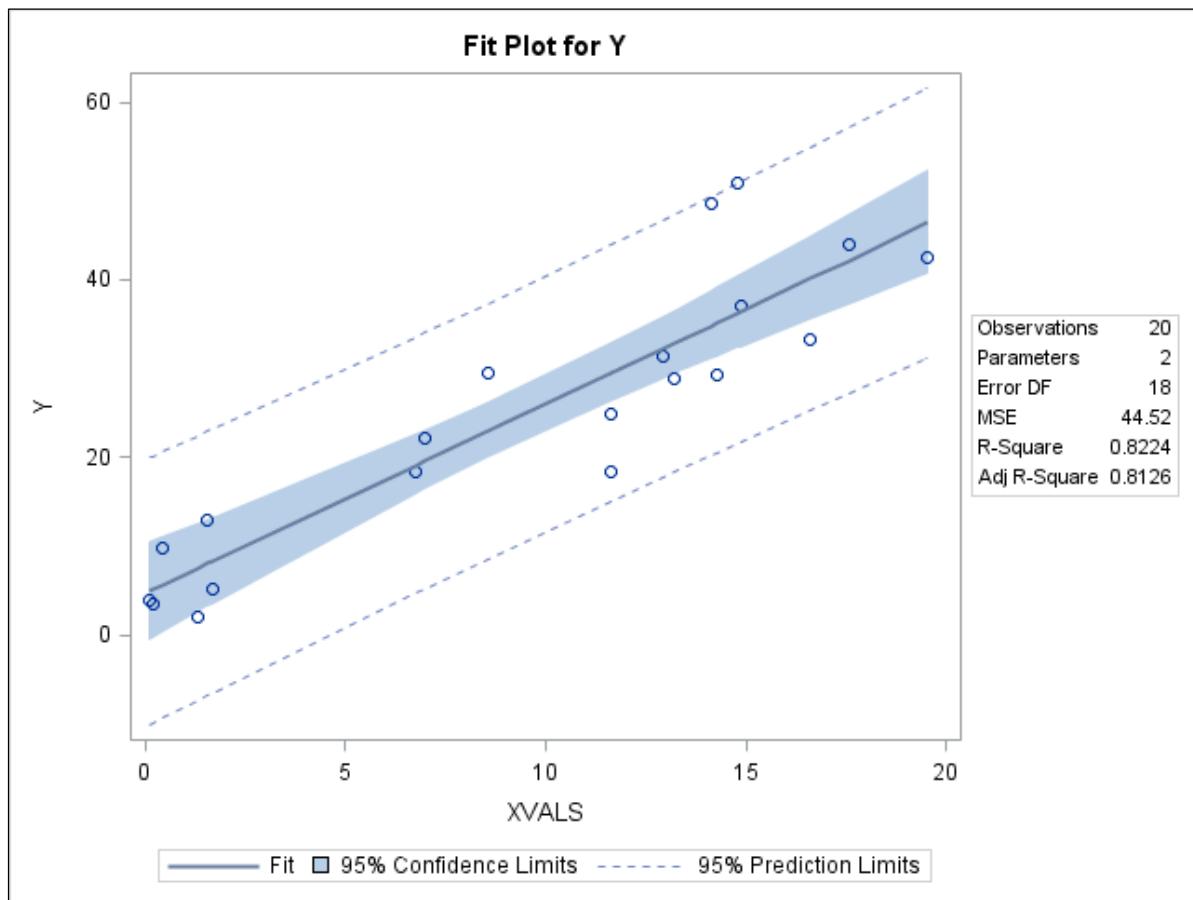
```
submit;
  ods select fitplot parameterestimates;
  proc reg data=sp4r.betaData;
    model y = xvals;
  run;
endsubmit;
quit;
```

```
The REG Procedure
Model: MODEL1
Dependent Variable: Y

Parameter Estimates

Variable DF Parameter Estimate Standard Error t Value Pr > |t|
Intercept 1 4.72359 2.66003 1.78 0.0927
XVALS 1 2.13263 0.23358 9.13 <.0001
```

The parameter estimates from PROC REG are identical to the estimates computed in IML.



End of Demonstration

7.4 Simulations

Objectives

- Use a DO or DO WHILE loop to create an iterative process.
- Use IF-THEN/ELSE statements to conduct conditional processing.
- Create an IML simulation without SAS procedures and with SAS procedures.

91

Motivation

Use Monte Carlo simulation in SAS/IML to

- obtain an approximate solution to a problem
- evaluate statistical methods.



92



Monte Carlo simulation is named after the Casino de Monte-Carlo in Monaco.

Duplicating the R Script

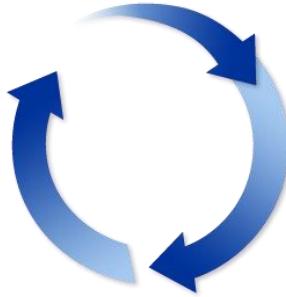
```
#Sampling distribution
simpleReg = function(n){

  beta0 = 5
  beta1 = 2
  xvals = runif(n)*n
  error = rnorm(n,mean=0,sd=5)
  y = beta0 + beta1*xvals + error

  x = cbind(rep(1,n),xvals)
  beta = solve(t(x)%*%x,t(x)%*%y)
  return(beta[2])
}

#Simulate
reps = 1000
samp = rep(0,reps)
for(i in 1:reps){
  samp[i] = simpleReg(20)
}

#Analyze
hist(samp,50)
mean(samp)
sd(samp)
```



93

Conditional Processing

Conditional processing statements used in a DATA step are identical to IML.

```
x = {1 2, 3 4};
miss = loc(x=.);
flag = isempty(miss);

if flag=0 then print x;
else print "empty matrix!";
```

*IF expression THEN statement;
<ELSE IF expression THEN statement;>
<ELSE statement;>*

94

As in a DATA step, the ELSE IF and ELSE statements are completely optional for conditional processing. In addition, the user can specify multiple ELSE IF statements.

Defining a DO Group

To execute multiple statements conditionally, use a DO statement.

```
X = {1 2, 3 4};
miss = loc(x=.);
flag = isempty(miss);

if flag=0 then do;
  print x;
  print "no missing elements!";
end;
else print "empty matrix!";
```

DO;
 statements;
END;

95

The DO statement specifies that the statements following the DO statement are executed as a group until a matching END statement appears.

DO statements often appear with clauses invoking iterative execution or in IF-THEN/ELSE statements so that the group of statements is executed only when the IF condition is satisfied.

Iterative DO Loops

The DO loop is identical to the **for(){}** loop in R.

```
for(i in 1:10){
  print(i)
}
```

```
do i=1 to 10 by 1;
  print i;
end;
```

DO start TO stop <BY increment>;
 statements;
END;

- The OUTPUT statement is used only in a DATA step.

96

SUBMIT Blocks with Loops and Conditions

- SUBMIT blocks can be combined with IF statements to execute SAS procedures and DATA steps conditionally.
- SUBMIT blocks can be combined with loops to execute SAS procedures and DATA steps repeatedly.

```
proc iml;
  do i = 1 to 1000;
    if i <= 500 then do;
      submit block;
    end;
    else do;
      submit block;
    end;
  end;
quit;
```

97

DO WHILE Loops

A DO WHILE statement duplicates the **while()** function in R.

The screenshot shows the SAS Studio interface with the R editor open. The code in the editor is:

```
x=1
while(x<5){
  print(x)
  x = x+1
}
```

```
x=1;
do while(x<5);
  print x;
  x = x+1;
end;
```

```
DO WHILE(expression);
  statements;
END;
```

98

Example: The Monty Hall Problem

- You are a guest on a game show.
- The host presents you with three doors.
- One door hides a car. The other two doors hide goats.



99

Example: The Monty Hall Problem

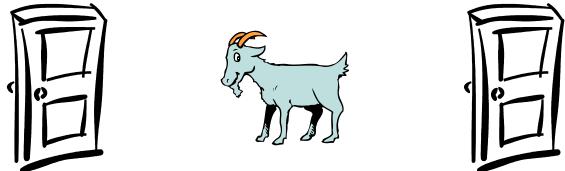
- You pick a door.



100

Example: The Monty Hall Problem

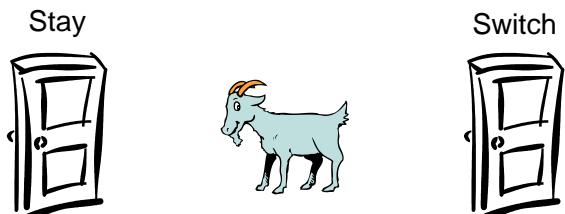
- The host (who knows what is behind each door) opens one of the two doors that you did not pick and reveals a goat.
- The host never opens the door picked by the contestant. The host always reveals a goat, never the car.



101

Example: The Monty Hall Problem

- The host then gives you the option of staying with your initial choice or switching to the remaining closed door.
- Which should you do?

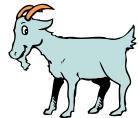


102

Example: The Monty Hall Problem

- Switching yields a $2/3$ chance of winning the car.
- Staying yields a $1/3$ chance of winning the car.

Stay



Switch





The Monty Hall Problem

SP4R07d06.sas

This demonstration shows how you might obtain empirical estimates of the likelihood of winning with each strategy (switch or stay) for the Monty Hall problem. The program in this demonstration is designed to be easy to follow. As a result, the program is not very efficient.

- Make your simulations more efficient by removing DO loops when possible.

 1. Let the number of simulations be 10,000 and set the random number seed to 802.

```
proc iml;
  numberIterations=10000;
  call randseed(802);
```

2. Start the simulation loop, which runs the number of times equal to **numberIterations**. The first step in simulating the Monty Hall problem is to choose which of the three doors hides the car. Use the SAMPLE function to draw a random door, {1 2 3}.

```
*Begin simulation;
do iteration=1 to numberIterations;
  doors = {1 2 3};
  carDoor=sample(doors,1);
```

3. For the sake of simplicity, always choose door 1. Monty Hall never opens the chosen door and never opens the door hiding the car. If the chosen door (door 1) hides the car, Monty randomly chooses between doors 2 and 3 (represented by a draw from a Bernoulli distribution with probability .5). If the car is hidden behind door 2, Monty Hall must open door 3. (He cannot open the door hiding the car or the door that you chose). If the car is hidden behind door 3, Monty Hall must open door 2.

```
*Pick door for Monty Hall to open;
if carDoor=1 then openDoor=randfun(1,"Bernoulli",.5) + 2;
else if carDoor=2 then openDoor=3;
else if carDoor=3 then openDoor=2;
```

4. Using a switching strategy requires switching to the unopened door that was not previously chosen. If Monty Hall opened door 2, switch to door 3. If Monty Hall opened door 3, switch to door 2.

```
*Determine door for switching strategy;
if openDoor=2 then switchDoor=3;
else if openDoor=3 then switchDoor=2;
```

5. If the car is behind door number 1, then the staying strategy wins because door number 1 was initially chosen. If the car is behind the door that would be chosen based on the switching strategy, then the switching strategy wins.

```
*Determine which strategy wins;
if carDoor=1 then stayWin=1;
else stayWin=0;

if carDoor=switchDoor then switchWin=1;
else switchWin=0;
/*switchWin=carDoor=switchDoor;*/
```

6. Append the results for the current iteration to a matrix called **results** and end the simulation loop.

```
*Collect results to a single matrix;
results=results // (iteration || carDoor || openDoor || stayWin
|| switchWin);
end;
```

7. Print the first 10 rows of the **results** matrix to show the outcome for every iteration. Calculate and print the percentage of iterations for which each strategy won.

```
reset noname;
resultsSubset = results[1:10,];
print resultsSubset [colname={iteration carDoor openDoor
stayWin switchWin}];

percentageWins=results[:,{4 5}];
print percentageWins [colname={stay switch}];
quit;
```

ITERATION	CARDOOR	OPOENDOOR	STAYWIN	SWITCHWIN
1	2	3	0	1
2	3	2	0	1
3	2	3	0	1
4	1	3	1	0
5	1	2	1	0
6	2	3	0	1
7	2	3	0	1
8	3	2	0	1
9	2	3	0	1
10	2	3	0	1

STAY	SWITCH
0.3297	0.6703

End of Demonstration

IML Simulations

There are three simulation methods in IML:

- simulate entirely in IML
 - ignores SAS procedures
- iteratively call SAS procedures
 - similar to using R functions
- use SAS procedures and a BY statement to avoid the DO loop



105

Analyzing Data with a BY Statement

- Analyzing each simulated data set one at a time is very inefficient.
- More efficient method:
 - Output all simulated data to a single SAS data set with a variable indicating the iteration number.
 - Analyze the data using a SAS procedure (for example, PROC GLM) with a BY statement, and output the results to a SAS data set.
 - Separate results are output for each iteration.

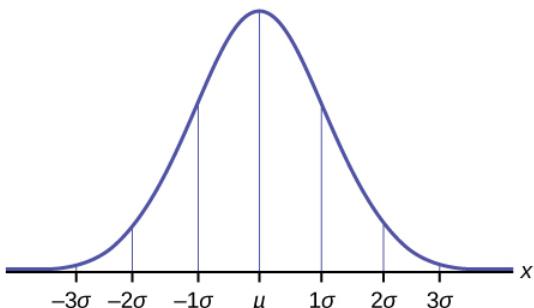


106

Example: Sampling Distribution

The sampling distribution is the probability distribution of a given statistic based on a random sample of size n .

- It can be considered as the distribution of the statistics for all possible samples from the same population of a given size.



107

Recall that the sampling distribution for the mean statistic of any distribution is normally distributed.

Example: Sampling Distribution

Use IML to identify the sampling distribution for the parameters of a simple linear regression model.

$$\hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{bmatrix} = (X^T X)^{-1} X^T Y$$

Find the sampling distribution using all three IML simulation methods.

108



Sampling Distribution – Method 1

SP4R07d07.sas

Compute the sampling distribution for simple linear regression parameters without SAS procedures.

1. Invoke PROC IML and create a subroutine to generate data from a simple linear regression model.

```
proc iml;
  start simpleRegFunc(n,beta0,beta1);
    xvals = randfun(n,"Uniform");
    xvals = xvals*20;
    error = randfun(n,"Normal",0,5);
    y = beta0 + beta1*xvals + error;
    x = j(n,1,1)||xvals;
    betaHat = inv(x`*x)*(x`*y);
    return(betaHat);
  finish;
```

The user-defined IML function SimpleRegFunc uses the input's sample size and the true values of β_0 and β_1 to generate values from a simple linear regression model $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, where $\varepsilon_i \sim N(0, \sigma^2 = 5)$, and the X values are uniformly generated between 0 and 20. The function returns the estimate $\hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{bmatrix}$.

2. The sample size for each iteration is set at 20, and 1,000 replications are conducted. The vectors **beta0** and **beta1** are initialized to save the estimates for each iteration.

```
n = 20;
reps = 1000;
beta0 = j(reps,1,.);
beta1 = j(reps,1,.);
call randseed(27606);
```

3. The simulation is conducted using a DO loop, and it uses true values of $\beta_0 = 5$ and $\beta_1 = 2$.

```
do i=1 to reps;
  betas = simpleRegFunc(n,5,2);
  beta0[i] = betas[1];
  beta1[i] = betas[2];
end;
```

4. The mean, standard deviation, 2.5% percentile, and 97.5% percentile are computed and printed in IML.

```

mean0 = mean(beta0);
sd0 = std(beta0);
call qntl(percentiles0,beta0,.025, .975);

mean1 = mean(beta1);
sd1 = std(beta1);
call qntl(percentiles1,beta1,.025, .975);

out0 = mean0//sd0//percentiles0;
reset noname;
print out0[colname="Beta0"
           rowname={"Mean","Standard Deviation","LCL","UCL"}];

out1 = mean1//sd1//percentiles1;
print out1[colname="Beta1"
           rowname={"Mean","Standard Deviation","LCL","UCL"}];
quit;

```

IML Output

Beta0	
Mean	4.9976255
Standard Deviation	2.389557
LCL	0.1748867
UCL	10.079698
Beta1	
Mean	2.0048417
Standard Deviation	0.2132723
LCL	1.5929447
UCL	2.4480762

End of Demonstration



Sampling Distribution – Method 2

SP4R07d08.sas

Compute the sampling distribution for simple linear regression parameters by iteratively calling SAS procedures.

1. Start by creating a subroutine to output generated data from a simple linear model.

```
proc iml;
  start simpleRegSub(xvals,yvals,n);
    beta0 = 5;
    beta1 = 2;
    xvals = randfun(n,"Uniform");
    xvals = xvals*20;
    error = randfun(n,"Normal",0,5);
    yvals = beta0 + beta1*xvals + error;
  finish;
```

The IML subroutine generates n (a user input value) values from a simple linear regression model $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, where $\beta_0 = 5$, $\beta_1 = 2$, $\varepsilon_i \sim N(0, \sigma=5)$, and the X values are uniformly generated between 0 and 20. The subroutine outputs only the response and independent variable values.

2. The sample size for each iteration is set at 20, and 1,000 replications are conducted. The vectors **beta0** and **beta1** are initialized to save the estimates for each iteration. The **time()** function is used to time the simulation.

```
n = 20;
reps = 1000;
beta0 = j(reps,1,.);
beta1 = j(reps,1,.);
call randseed(27606);
startTime = time();
```

3. Each iteration of the simulation calls the SimpleRegSub subroutine. Then the response and independent variables are passed to a SAS data set using the CREATE, APPEND, and CLOSE statements.

```
do i=1 to reps;
  call simpleRegSub(x,y,n);

  create sp4r.simulation var {x y};
  append;
  close sp4r.simulation;
```

4. For each newly created SAS data set, PROC REG is used to output the model parameters estimates with the ODS OUTPUT statement. The ODS SELECT NONE statement withholds all SAS procedure output. Printing can be time consuming and highly inefficient in a simulation. The ODS SELECT DEFAULT statement turns off the ODS SELECT NONE statement.

```

submit;
  ods select none;
  ods output ParameterEstimates=sp4r.params;
  proc reg data=sp4r.simulation;
    model y=x;
  run;
  ods select default;
endsubmit;

```

5. The parameter estimates are passed back into IML using the USE, READ, and CLOSE statements. The iteration's parameter estimates are stored in the **beta0** and **beta1** vectors.

```

use sp4r.params;
read all var {estimate} into estimates;
close sp4r.params;

beta0[i]=estimates[1];
beta1[i]=estimates[2];
end;

```

6. The mean, standard deviation, 2.5% percentile, and 97.5% percentile are computed and printed in IML.

```

mean0 = mean(beta0);
sd0 = std(beta0);
call qntl(percentiles0,beta0,.025, .975);

mean1 = mean(beta1);
sd1 = std(beta1);
call qntl(percentiles1,beta1,.025, .975);

out0 = mean0//sd0//percentiles0;
reset noname;
print out0[colname="Beta0"
  rowname={"Mean","Standard Deviation","LCL","UCL"}];

out1 = mean1//sd1//percentiles1;
print out1[colname="Beta1"
  rowname={"Mean","Standard Deviation","LCL","UCL"}];

total = time() - startTime;
print total[colname="Elapsed Time"];
quit;

```

IML Output

	Beta0
Mean	4.9976255
Standard Deviation	2.389557
LCL	0.1748867
UCL	10.079698
	Beta1
Mean	2.0048417
Standard Deviation	0.2132723
LCL	1.5929447
UCL	2.4480762
	Elapsed Time
	31.993

End of Demonstration



Sampling Distribution – Method 3

SP4R07d09.sas

Compute the sampling distribution for simple linear regression parameters using a BY statement.

- The simulation uses a sample size of 20 and conducts 10,000 replications.

```
proc iml;
  startTime = time();
  simulations = 1000;
  sampleSize = 20;
```

- The **simulationNumber** vector is simply a sequence from 1 to 1,000. The **each** vector creates a vector that is the same size with each entry corresponding to the sample size. Finally, the REPEAT function is used to duplicate each simulation number (1 to 1,000) 20 times.

```
simulationNumber = 1:simulations;
each = j(simulations,1,sampleSize);
simulationNumber = repeat(simulationNumber,each)`;
```

- The IML code generates a total of $20 \times 1,000 = 20,000$ values from a simple linear regression model $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, where $\beta_0 = 5$, $\beta_1 = 2$, $\varepsilon_i \sim N(0, \sigma^2 = 5)$, and the X values are uniformly generated between 0 and 20.

```
call randseed(27606);
total = simulations*sampleSize;
beta0 = 5;
beta1 = 2;
xvals = randfun(total,"Uniform");
x = xvals*20;
error = randfun(total,"Normal",0,5);
y = beta0 + beta1*x + error;
```

- Pass the response and independent variables to a SAS data set along with the simulation number.

```
create sp4r.simulation var {simulationNumber x y};
append;
close sp4r.simulation;
```

- Use PROC REG to output the parameter estimates for the model by using the ODS SELECT statement. In addition, use the BY statement to compute parameter estimates for each simulation number. This results in 1,000 parameter estimates being saved in the **params** data set.

```
submit;
  ods select none;
  ods output ParameterEstimates=sp4r.params;
  proc reg data=sp4r.simulation;
    by simulationNumber;
    model y=x;
  run;
  ods select default;
endsubmit;
```

- Read in the estimates for $\hat{\beta}_0$ and $\hat{\beta}_1$ separately by using the WHERE option. The estimates in the **params** data set are called **estimate** and the parameters are referred to as 'Intercept' and 'X'.

```

use sp4r.params;
read all var {estimate} where (variable='Intercept') into beta0;
close sp4r.params;

use sp4r.params;
read all var {estimate} where (variable='X') into beta1;
close sp4r.params;

```

7. The mean, standard deviation, 2.5% percentile, and 97.5% percentile are computed and printed in IML.

```

mean0 = mean(beta0);
sd0 = std(beta0);
call qntl(percentiles0,beta0,.025, .975);

mean1 = mean(beta1);
sd1 = std(beta1);
call qntl(percentiles1,beta1,.025, .975);

out0 = mean0//sd0//percentiles0;
reset noname;
print out0[colname="Beta0"
           rowname={"Mean", "Standard Deviation", "LCL", "UCL"}];

out1 = mean1//sd1//percentiles1;
print out1[colname="Beta1"
           rowname={"Mean", "Standard Deviation", "LCL", "UCL"}];

total = time() - startTime;
print total[colname="Elapsed Time"];
quit;

```

IML Output:

Beta0	
Mean	4.9996746
Standard Deviation	2.2283769
LCL	0.8000621
UCL	9.4774217

Beta1	
Mean	2.0002625
Standard Deviation	0.1966287
LCL	1.5954792
UCL	2.3794159

Elapsed Time
3.1209998

End of Demonstration



Exercises

2. Generating a Multiple Regression Data Matrix and Computing Parameter Estimates

This exercise extends the ideas from the previous demonstration.

- a. Generate data from a multiple regression model, $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \varepsilon_i$, with 20 samples where $\beta_0 = 3$, $\beta_1 = 2$, $\beta_2 = -1$, and $\varepsilon_i \sim N(0, \sigma=5)$. Let $x_{1i} \sim Uniform(0, 20)$ and $x_{2i} \sim Uniform(10, 30)$. Use the seed 27606 to duplicate your results. Generate the random numbers using the RANDFUN function. Print the generated values.

y	beta0	beta1	beta2	xvals1	xvals2	error
-17.0778	3	2	-1	6.9833149	28.419994	-5.624434
20.399422				11.619663	13.11864	7.2787367
6.9089148				14.863959	25.720417	-0.098586
27.837334				17.563605	14.31566	4.025783
-23.17483				0.0771578	24.440131	-1.889011
6.8605901				16.60792	26.879481	-2.475768
17.609951				14.251058	18.034902	4.1427361
22.060529				14.773279	12.525219	2.0391891
-17.91391				1.6650986	29.064125	4.8200187
2.5788287				11.628301	10.744335	-12.93344
7.0012405				14.12347	19.860202	-4.385498
-17.41216				1.5470275	20.552019	-2.954198
20.68621				19.547	25.236564	3.8287744
1.3362742				12.918223	29.589143	2.0889713
-7.890633				1.3057261	18.96901	5.4669255
6.5089801				8.558932	16.789311	3.1804269
8.1659533				6.7389815	20.715758	12.403748
-24.19493				0.425611	20.222944	-7.823206
-1.814433				0.1731237	10.720545	5.5598646
9.0943547				13.190648	22.985184	2.6982433

- b. Create the design matrix and compute $\hat{\beta} = (X^T X)^{-1} X^T Y$ using the INV function. Print your results.

```
x
1 6.9833149 28.419994
1 11.619663 13.11864
1 14.863959 25.720417
1 17.563605 14.31566
1 0.0771578 24.440131
1 16.60792 26.879481
1 14.251058 18.034902
1 14.773279 12.525219
1 1.6650986 29.064125
1 11.628301 10.744335
1 14.12347 19.860202
1 1.5470275 20.552019
1 19.547 25.236564
1 12.918223 29.589143
1 1.3057261 18.96901
1 8.5558932 16.789311
1 6.7389815 20.715758
1 0.425611 20.222944
1 0.1731237 10.720545
1 13.190648 22.985184

betaHat
4.5097506
2.0297133
-1.040226
```

- c. Compute and print the estimates $\hat{\sigma}^2$ and $\hat{\sigma}$ where $\hat{\sigma}^2 = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n-1}$. Recall that SAS does not use the $^{\wedge}$ operator to exponentiate matrix elements.

```
sigma2Hat sigmaHat
33.790635 5.8129713
```

3. Creating User-Defined Functions and Subroutines

Standardized values are computed as $\frac{x - \bar{x}}{std(x)}$, where $std(x) = \sqrt{\frac{\sum(x - \bar{x})^2}{n-1}}$.

- a. Create a function, STANDARDIZE, that takes a matrix as an input and returns the matrix with each column standardized.
- b. Create a 10×3 matrix of random numbers where the first column is $N(5,5)$, the second column is *Uniform(10,15)*, and the third column is *Exponential(7)*. Use the seed 802 to duplicate your results. Print the matrix and then use the STANDARDIZE function to create and print the standardized matrix.

```

mymat

11.606944 11.863687 18.710933
5.2331384 13.862579 8.2257589
5.0699716 13.325589 13.889108
9.2352147 12.471574 4.3133361
7.5553052 13.356539 1.1156816
6.7021899 12.096431 14.181072
-1.018837 14.853318 1.4487006
6.0402131 13.289878 10.497412
2.9340508 12.488342 1.2435366
11.963293 12.219451 4.9053748

stand

1.297565 -1.205853 1.7341685
-0.332141 0.9480863 0.0595209
-0.373861 0.369444 0.964047
0.691142 -0.550814 -0.565355
0.2616091 0.4027946 -1.076071
0.0434776 -0.955056 1.0106784
-1.930697 2.0156739 -1.022882
-0.125782 0.3309627 0.4223396
-0.919991 -0.532745 -1.05565
1.3886794 -0.822494 -0.470797

```

- c. Alter the STANDARDIZE function and create the subroutine STANDSUB. Let the subroutine take a matrix as input and output the standardized matrix, as well as the column means and standard deviations.
- d. Generate the same data matrix and use the subroutine to create and print the three matrices.

```

m

6.5321484 12.982739 7.8530913

s

3.9110141 0.9280167 6.2611224

standardized

1.297565 -1.205853 1.7341685
-0.332141 0.9480863 0.0595209
-0.373861 0.369444 0.964047
0.691142 -0.550814 -0.565355
0.2616091 0.4027946 -1.076071
0.0434776 -0.955056 1.0106784
-1.930697 2.0156739 -1.022882
-0.125782 0.3309627 0.4223396
-0.919991 -0.532745 -1.05565
1.3886794 -0.822494 -0.470797

```

- #### 4. Using a SAS Data Set, Creating an IML Module, and Exporting Results to a New Data Table
- a. Print the **govtDemand** data set and notice that each continuous variable has missing values.
 - b. Read the **govtDemand** data set into an IML matrix named **govt**.

- c. Create a function that takes a vector as input and imputes all missing values with the mean of the vector and returns the imputed vector.
 - d. Impute columns 2 through 4 and create a new SAS data set named **govtImputed**, with the same names as **govtDemand**, that contains the imputed matrix. Because a matrix is being exported to a SAS data set, be sure to use the COLNAME= option in the CREATE statement.
 - e. Finally, print the SAS data set. Also use the SUBMIT block to run PROC CORR on the variables **agric**, **manu**, and **labor**.

Obs	YEAR	AGRIC	MANU	LABOR
1	1982	600.00	1000.00	600.00
2	1983	1100.00	1200.00	792.00
3	1984	1100.00	1350.00	800.00
4	1985	1150.00	1547.31	825.00
5	1986	1200.00	1475.00	850.00
6	1987	1272.92	1500.00	900.00
7	1988	1400.00	1650.00	920.00
8	1989	1420.00	1650.00	886.69
9	1990	1272.92	1680.00	940.00
10	1991	1450.00	1700.00	950.00
11	1992	1450.00	1720.00	975.00
12	1993	1460.00	1720.00	975.00
13	1994	1470.00	1730.00	1000.00
14	1995	1475.00	1740.00	1000.00

The CORR Procedure

Simple Statistics						
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
AGRIC	14	1273	239.90750	17821	600.00000	1475
MANU	14	1547	225.10353	21662	1000	1740
LABOR	14	886.69231	108.56999	12414	600.00000	1000

Pearson Correlation Coefficients, N = 14
 Prob > |r| under H0: Rho=0

	AGRIC	MANU	LABOR
AGRIC	1.00000	0.94119	0.96682
	<.0001	<.0001	
MANU	0.94119	1.00000	0.95092
	<.0001		<.0001
LABOR	0.96682	0.95092	1.00000
	<.0001	<.0001	

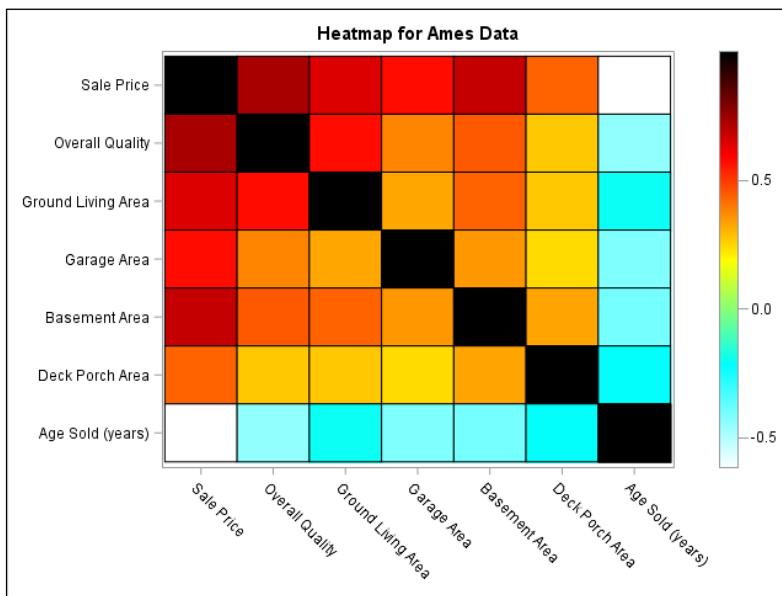
5. Calling Statistical Graphics from SAS/IML

- a. Read the variables `saleprice`, `overall_qual`, `gr_liv_area`, `garage_area`, `basement_area`, `deck_porch_area`, and `age_sold` from the `AmesHousing` data set into an IML matrix named `imlAmes`.

- b. Create a correlation matrix from **imlAmes** named **corrAmes** and print it.

corrAmes							
1	0.7345057	0.6504636	0.5789207	0.6895635	0.439889	-0.615425	
0.7345057	1	0.5787329	0.3859067	0.4564424	0.2795069	-0.442376	
0.6504636	0.5787329	1	0.3328336	0.4398542	0.2805839	-0.192722	
0.5789207	0.3859067	0.3328336	1	0.3562982	0.2498748	-0.413458	
0.6895635	0.4564424	0.4398542	0.3562982	1	0.3368862	-0.39529	
0.439889	0.2795069	0.2805839	0.2498748	0.3368862	1	-0.205836	
-0.615425	-0.442376	-0.192722	-0.413458	-0.39529	-0.205836	1	

- c. Navigate to the SAS/IML documentation and review the HEATMAP subroutine. Create a heat map of the correlation matrix. Use the XVALUES= and YVALUES= options to set appropriate labels for the rows and columns of the plot. Also, provide the map with a title. Finally, change the color coding of the heat map to "Temperature".



- d. Go to the **Work** directory and open the **_heatmap** data set. SAS/IML exported the data set that is required to be used by the SGLOT procedure to create the heat map.

6. Simulating the Birthday Problem

Use simulation to calculate the empirical probability of two people sharing the same birthdate in a group of 23 people. Use 1000 iterations. Assume that none of the people is born on Leap Day and every birthdate is equally likely.

- Invoke PROC IML, set the random seed, and begin a DO loop with 1000 simulations. Create a vector named **pair** to hold the results of each iteration.
- Draw 23 birthdates using the SAMPLE function. (Dates can be represented as the numbers 1 through 365.)
- Check whether any two birthdates are the same. (Hint: Use the UNIQUE function.)
- If at least two birthdates are the same, set the variable **pair** to 1. Otherwise, set **pair** to zero.
- Calculate the proportion of iterations in which a pair was found.

proportion

0.506

- f. How can you avoid using the DO loop for this simulation?

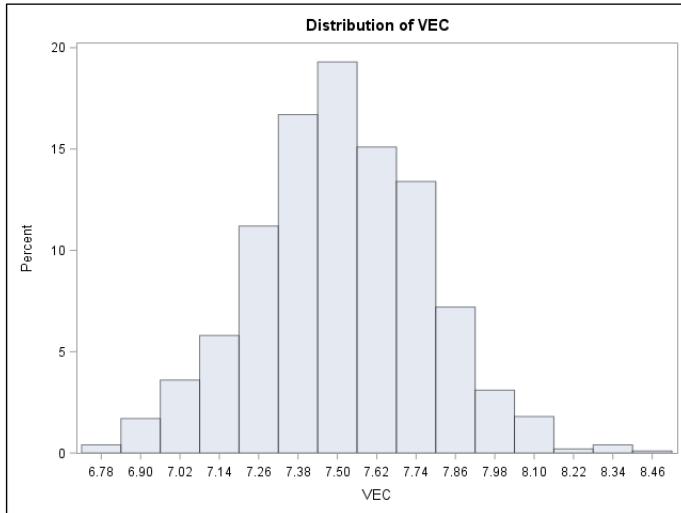
7. Using IML to Identify the Sampling Distribution for a Uniform Random Variable

- Invoke IML and create a function with inputs, n (for sample size), and a minimum and maximum value. Within the user-defined function create a vector with n random numbers from the distribution $\text{Uniform}(\min, \max)$.
- Set the sample size to 30, the minimum to 5, and the maximum to 10. Set the number of replications to 1,000 and the seed to 802. Finally, initialize a vector to store the mean of each simulation value.
- Begin a DO loop and use the user-defined function to compute and store the mean of each sample for each iteration.
- Pass the simulation results to a SAS data set.
- Submit PROC UNIVARIATE code to compute the BasicMeasures table and a histogram of the simulation results.

Basic Statistical Measures

Location	Variability
----------	-------------

Mean	7.509750	Std Deviation	0.26461
Median	7.512882	Variance	0.07002
Mode	.	Range	1.72576
		Interquartile Range	0.35543



8. Redoing the Previous Exercise Using a BY Statement and Avoiding the DO Loop

- Invoke PROC IML and create a variable corresponding to 1,000 simulations, a sample size of 30, a minimum value of 5, and a maximum value of 10.
- Create a vector that represents the simulation number for each sampled data point. Let each simulation number repeat 30 times. (Hint: Use the REPEAT function.)

- c. Generate data from a uniform random variable with minimum and maximum values of 5 and 10 respectively. Notice that the total number of observations should be the number of simulations times the sample size (1,000*30=30,000).
- d. Pass the vector of random numbers and the simulation number to a SAS data set.
- e. Use an appropriate procedure to output the mean of each sample from each simulation using a BY statement. Be sure to use the ODS SELECT NONE and ODS SELECT DEFAULT statements.
- f. Read the means back into PROC IML.
- g. Compute and print the mean, standard deviation, 2.5% percentile, and the 97.5% percentile for the means data.

Mu	
Mean	7.5097502
Standard Deviation	0.2646051
LCL	6.9767451
UCL	8.0408154

End of Exercises

7.5 Solutions

Solutions to Exercises

1. Practicing with Basic Operations

In this exercise, you perform operations on the data used in the previous demonstration. Use the code below at the beginning of the exercise program.

```
proc iml;
  items      ={'Groceries','Utilities','Rent','Car Expenses',
                'Fun Money','Personal Expenses'};
  weeks      ={'Week 1','Week 2','Week 3','Week 4'};
  amounts    ={96 78 82 93,
              61 77 62 68,
              300 300 300 300,
              25 27 98 18,
              55 34 16 53,
              110 85 96 118};
  weeklyIncome ={900 850 1050 950};
  weeklyExpenses=amounts[+,];
```

- a. Create a 1×4 matrix named **proportionIncomeSpent** whose elements are the proportion of each week's income that went to expenses. Use the RESET statement to suppress the automatic printing of matrix names. Print the **proportionIncomeSpent** matrix with the values of **weeks** used as column labels and PERCENT7.2 used as a format.

```
proportionIncomeSpent=weeklyExpenses / weeklyIncome;
reset noname;
print "Proportion of income spent each week",
      proportionIncomeSpent[colname=weeks format=percent7.2];
```

PROC IML Output

Proportion of income spent each week			
	Week 1	Week 2	Week 3
	71.9%	70.7%	62.3%
			68.4%

- b. Create a 1×4 matrix named **proportionIncomeSaved** whose elements are equal to the proportion of each week's income that did not go to expenses. That is, use an implicit loop to subtract the values of **proportionIncomeSpent** from one. Print the **proportionIncomeSaved** matrix with the values of **weeks** used as column labels and PERCENT7.2 used as a format.

```
proportionIncomeSaved=1 - proportionIncomeSpent;
print "Proportion of income saved each week",
      proportionIncomeSaved[colname=weeks format=percent7.2];
```

PROC IML Output

Proportion of income saved each week			
	Week 1	Week 2	Week 3
	28.1%	29.3%	37.7%
			31.6%

- c. Create a 6×4 matrix named **proportionSpentPerItem** whose elements are the proportion of each week's income spent on each item, by week. That is, use an implicit loop to divide the **amounts** matrix by the **weeklyIncome** matrix. Print the **proportionSpentPerItem** matrix with the values of **items** used as row labels, the values of **weeks** used as column labels, and PERCENT7.2 used as a format.

```
proportionSpentPerItem=amounts/weeklyIncome;
print "Percentage of income spent on each item, by week",
      proportionSpentPerItem [rowname=items
                                colname=weeks format=percent7.2];
```

	Percentage of income spent on each item, by week			
	Week 1	Week 2	Week 3	Week 4
Groceries	10.7%	9.18%	7.81%	9.79%
Utilities	6.78%	9.06%	5.90%	7.16%
Rent	33.3%	35.3%	28.6%	31.6%
Car Expenses	2.78%	3.18%	9.33%	1.89%
Fun Money	6.11%	4.00%	1.52%	5.58%
Personal Expenses	12.2%	10.0%	9.14%	12.4%

- d. Create a matrix named **weeklyExpenseChange** with the same number of rows as **amounts** but with one less column than **amounts** (in other words, a 6×3 matrix). Fill it with missing numeric values. This should be done with a matrix literal. Fill each column of **weeklyExpenseChange** with each column of **amounts** minus the previous column of **amounts**. That is, column 1 of **weeklyExpenseChange** should equal column 2 of **amounts** minus column 1 of **amounts**, and so on. Print a title and print the matrix. Use columns 2 through 4 of **weeks** as column labels and use **items** as row labels.

```
weeklyExpenseChange={. . .,
                     . . .,
                     . . .,
                     . . .,
                     . . .,
                     . . .};

weeklyExpenseChange [,1]=amounts[,2] - amounts[,1];
weeklyExpenseChange [,2]=amounts[,3] - amounts[,2];
weeklyExpenseChange [,3]=amounts[,4] - amounts[,3];

print "Change in spending from previous week, by item",
      weeklyExpenseChange [rowname=items
                            colname={"Week 2", "Week 3", "Week 4"}];
quit;
```

	Change in spending from previous week, by item		
	Week 2	Week 3	Week 4
Groceries	-18	4	11
Utilities	16	-15	6
Rent	0	0	0
Car Expenses	2	71	-80
Fun Money	-21	-18	37
Personal Expenses	-25	11	22

2. Generating a Multiple Regression Data Matrix and Computing Parameter Estimates

This exercise extends the ideas from the previous demonstration.

- a. Generate data from a multiple regression model, $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \varepsilon_i$, with 20 samples where $\beta_0 = 3$, $\beta_1 = 2$, $\beta_2 = -1$, and $\varepsilon_i \sim N(0, \sigma^2 = 5)$. Let $x_{1i} \sim Uniform(0, 20)$ and $x_{2i} \sim Uniform(10, 30)$. Use the seed 27606 to duplicate your results. Generate the random numbers using the RANDFUN function. Print the generated values.

```
proc iml;
  call randseed(27606);
  n = 20;
  beta0 = 3;
  beta1 = 2;
  beta2 = -1;
  xvals1 = randfun(n,"Uniform");
  xvals1 = xvals1*20;
  xvals2 = randfun(n,"Uniform");
  xvals2 = (xvals2*20) + 10;
  error = randfun(n,"Normal",0,5);
  y = beta0 + beta1*xvals1 + beta2*xvals2 + error;
  print y beta0 beta1 beta2 xvals1 xvals2 error;
```

y	beta0	beta1	beta2	xvals1	xvals2	error
2.3890519	3	2	-1	15.535752	25.086444	-6.596008
27.055062				19.109842	16.336432	2.1718106
-4.292507				8.7436082	21.76381	-3.015913
11.835473				12.034105	20.732286	5.4995483
18.823426				12.89481	16.240074	6.2738806
-26.1453				1.948081	23.511204	-9.530258
27.55394				19.206666	15.207025	1.3476321
15.761662				14.474217	17.647193	1.460422
12.815418				12.986986	18.152577	1.9940234
19.421855				18.214594	25.124106	5.1167724
16.696806				15.184676	29.289954	12.617408
-15.49262				2.9719555	27.424333	2.9878057
6.7790585				15.249747	20.255948	-6.464489
31.024149				19.654065	11.005698	-0.278282
6.8295732				9.9692864	27.152395	11.043396
-8.10431				0.0501745	11.999825	0.795166
-10.70045				1.934529	16.587159	-0.982352
28.096877				19.974169	10.011983	-4.839478
5.1143731				9.217379	13.189886	-3.130499
6.3452045				16.397849	27.952028	-1.498465

- b. Create the design matrix and compute $\hat{\beta} = (X^T X)^{-1} X^T Y$ using the INV function. Print your results.

```
x = j(n,1,1)||xvals1||xvals2;
betaHat = inv(x`*x)*(x`*y);
print x, betaHat;
*Alternative SAS Function;
*betaHat = solve( (x`*x)*(x`*y) );
*print betaHat;
```

```

x

1 6.9833149 28.419994
1 11.619663 13.11864
1 14.863959 25.720417
1 17.563605 14.31566
1 0.0771578 24.440131
1 16.60792 26.879481
1 14.251058 18.034902
1 14.773279 12.525219
1 1.6650986 29.064125
1 11.628301 10.744335
1 14.12347 19.860202
1 1.5470275 20.552019
1 19.547 25.236564
1 12.918223 29.589143
1 1.3057261 18.96901
1 8.558932 16.789311
1 6.7389815 20.715758
1 0.425611 20.222944
1 0.1731237 10.720545
1 13.190648 22.985184

betaHat

4.5097506
2.0297133
-1.040226

```

- c. Compute and print the estimates $\hat{\sigma}^2$ and $\hat{\sigma}$ where $\hat{\sigma}^2 = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n-1}$. Recall that SAS does not use the $^{\wedge}$ operator to exponentiate matrix elements.

```

pred = x*betaHat;
sse = sum( (y-pred)##2 );
sigma2Hat = sse / (n-1);
sigmaHat = sqrt(sigma2Hat);
print sigma2Hat sigmaHat;
quit;

```

```

sigma2Hat  sigmaHat
33.790635 5.8129713

```

3. Creating User-Defined Functions and Subroutines

Standardized values are computed as $\frac{x - \bar{x}}{std(x)}$, where $std(x) = \sqrt{\frac{\sum(x - \bar{x})^2}{n-1}}$.

- a. Create a function, STANDARDIZE, that takes a matrix as an input and returns the matrix with each column standardized.

```

proc iml;
  start standardize(x);
    n=nrow(x);
    mean=x[:,];           /* means for all columns      */
    xbar=repeat(mean,n,1); /* n rows of means          */
    x=x-xbar;             /* center x to mean zero    */
    stdv=std(x);          /* standard deviations for columns */
    x=x/stdv;             /* scale to std dev 1        */
    return(x);
  finish;

```

The mean of each column here is computed using the reduction operator [:,].

- b. Create a 10 x 3 matrix of random numbers where the first column is 123, the second column is 123, and the third column is 123. Use the seed 802 to duplicate your results. Print the matrix and then use the STANDARDIZE function to create and print the standardized matrix.

```

n = 10;
call randseed(802);
mymat = randfun(n,"Normal",5,5)
  ||randfun(n,"Uniform",10,15)||randfun(n,"Exponential",7);
print mymat;
stand = standardize(mymat);
print stand;
quit;

```

```

mymat

11.606944 11.863687 18.710933
5.2331384 13.862579 8.2257589
5.0699716 13.325589 13.889108
9.2352147 12.471574 4.3133361
7.5553052 13.356539 1.1156816
6.7021899 12.096431 14.181072
-1.018837 14.853318 1.4487006
6.0402131 13.289878 10.497412
2.9340508 12.488342 1.2435366
11.963293 12.219451 4.9053748

```

```

stand

1.297565 -1.205853 1.7341685
-0.332141 0.9480863 0.0595209
-0.373861 0.369444 0.964047
0.691142 -0.550814 -0.565355
0.2616091 0.4027946 -1.076071
0.0434776 -0.955056 1.0106784
-1.930697 2.0156739 -1.022882
-0.125782 0.3309627 0.4223396
-0.919991 -0.532745 -1.05565
1.3886794 -0.822494 -0.470797

```

- c. Alter the STANDARDIZE function and create the subroutine STANDSUB. Let the subroutine take a matrix as input and output the standardized matrix, as well as the column means and standard deviations.

```
proc iml;
  start standsub(stand,mean,stdv,x);
    n=nrow(x);
    mean=x[:,];           /* means for all columns      */
    xbar=repeat(mean,n,1); /* n rows of means          */
    x=x-xbar;             /* center x to mean zero    */
    stdv=std(x);          /* standard deviations for columns */
    stand=x/stdv;         /* scale to std dev 1        */
  finish;
```

- d. Generate the same data matrix and use the subroutine to create and print the three matrices.

```
n = 10;
call randseed(802);
mymat = randfun(n,"Normal",5,5)
  ||randfun(n,"Uniform",10,15)||randfun(n,"Exponential",7);
call standsub(standardized,m,s,mymat);
print m, s, standardized;
quit;
```

m	
6.5321484 12.982739 7.8530913	
s	
3.9110141 0.9280167 6.2611224	
standardized	
1.297565 -1.205853 1.7341685 -0.332141 0.9480863 0.0595209 -0.373861 0.369444 0.964047 0.691142 -0.550814 -0.565355 0.2616091 0.4027946 -1.076071 0.0434776 -0.955056 1.0106784 -1.930697 2.0156739 -1.022882 -0.125782 0.3309627 0.4223396 -0.919991 -0.532745 -1.05565 1.3886794 -0.822494 -0.470797	

4. Using a SAS Data Set, Creating an IML Module, and Exporting Results to a New Data Table

- a. Print the **govtDemand** data set and notice that each continuous variable has missing values.

```
proc print data=sp4r.govtDemand;
run;
```

Obs	year	agric	manu	labor
1	1982	600	1000	600
2	1983	1100	1200	792
3	1984	1100	1350	800
4	1985	1150	.	825
5	1986	1200	1475	850
6	1987	.	1500	900
7	1988	1400	1650	920
8	1989	1420	1650	.
9	1990	.	1680	940
10	1991	1450	1700	950
11	1992	1450	1720	975
12	1993	1460	1720	975
13	1994	1470	1730	1000
14	1995	1475	1740	1000

- b. Read the **govtDemand** data set into an IML matrix named **govt**.

```
proc iml;
  use sp4r.govtDemand;
  read all into govt;
  close sp4r.govtDemand;
```

- c. Create a function that takes a vector as input and imputes all missing values with the mean of the vector and returns the imputed vector.

```
start impute(colvec);
  colvec[loc(colvec=.)] = mean(colvec);
  return(colvec);
finish impute;
```

The LOC function is to find the index of all missing values in the vector.

- d. Impute columns 2 through 4 and create a new SAS data set named **govtImputed**, with the same names as **govtDemand**, which contains the imputed matrix. Because a matrix is being exported to a SAS data set, be sure to use the COLNAME= option in the CREATE statement.

```
govtImputed = govt[,1]||impute(govt[,2])
  ||impute(govt[,3])||impute(govt[,4]);
create sp4r.newGovt from govtImputed
  [colname={year, agric, manu, labor}];
append from govtImputed;
close sp4r.newGovt;
```

- e. Finally, print the SAS data set and also run PROC CORR on the variables **agric**, **manu**, and **labor**.

```
submit;
  proc print data=sp4r.newGovt;run;
  proc corr data=sp4r.newGovt;
    var agric manu labor;
  run;
endsubmit;
quit;
```

The CORR Procedure						
3 Variables: AGRIC MANU LABOR			Simple Statistics			
Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
AGRIC	14	1273	239.90750	17821	600.00000	1475
MANU	14	1547	225.10353	21662	1000	1740
LABOR	14	886.69231	108.56999	12414	600.00000	1000
Pearson Correlation Coefficients, N = 14						
Prob > r under H0: Rho=0						
		AGRIC	MANU	LABOR		
		AGRIC	1.00000 <.0001	0.94119 <.0001	0.96682 <.0001	
		MANU	0.94119 <.0001	1.00000 <.0001	0.95092 <.0001	
		LABOR	0.96682 <.0001	0.95092 <.0001	1.00000 <.0001	

5. Calling Statistical Graphics from SAS/IML

- a. Read the variables `saleprice`, `overall_qual`, `gr_liv_area`, `garage_area`, `basement_area`, `deck_porch_area`, and `age_sold` from the `AmesHousing` data set into an IML matrix named `imlAmes`.

```
proc iml;
  use sp4r.ameshousing;
  read all var {saleprice overall_qual gr_liv_area garage_area
    basement_area deck_porch_area age_sold} into imlAmes;
  close sp4r.ameshousing;
```

- b. Create a correlation matrix from `imlAmes` named `corrAmes` and print it.

```
corrAmes = corr(imlAmes);
print corrAmes;
```

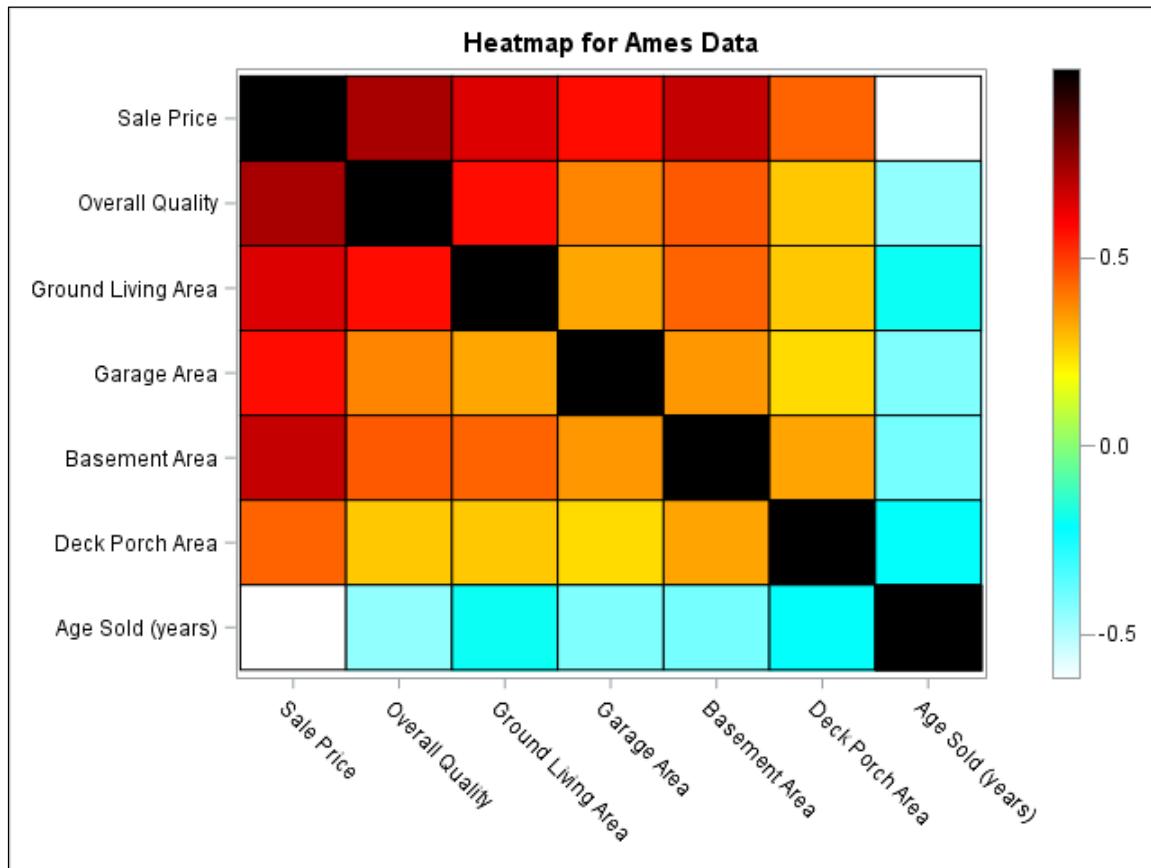
corrAmes						
1	0.7345057	0.6504636	0.5789207	0.6895635	0.439889	-0.615425
0.7345057	1	0.5787329	0.3859067	0.4564424	0.2795069	-0.442376
0.6504636	0.5787329	1	0.3328336	0.4398542	0.2805839	-0.192722
0.5789207	0.3859067	0.3328336	1	0.3562982	0.2498748	-0.413458
0.6895635	0.4564424	0.4398542	0.3562982	1	0.3368862	-0.39529
0.439889	0.2795069	0.2805839	0.2498748	0.3368862	1	-0.205836
-0.615425	-0.442376	-0.192722	-0.413458	-0.39529	-0.205836	1

- c. Navigate to the SAS/IML documentation and review the `HEATMAP` subroutine. Create a heat map of the correlation matrix. Use the `XVALUES=` and `YVALUES=` options to set appropriate labels for the rows and columns of the plot. Also, provide the map with a title. Finally, change the color coding of the heat map to "Temperature".

```

varNames = {"Sale Price" "Overall Quality" "Ground Living Area"
"Garage Area" "Basement Area" "Deck Porch Area"
"Age Sold (years)" };
call heatmapcont(corrAmes) xvalues=varNames yvalues=varNames
colorramp="Temperature" title="Heatmap for Ames Data";
quit;

```



- d. Go to the **Work** directory and open the **_heatmap** data set. SAS/IML exported the data set required to be used by the SGLOT procedure to create the heat map.

6. Simulating the Birthday Problem

Use simulation to calculate the empirical probability of two people sharing the same birthdate in a group of 23 people. Use 1000 iterations. Assume that none of the people is born on Leap Day and every birthdate is equally likely.

- a. Invoke PROC IML, set the random seed, and begin a DO loop with 1000 simulations. Create a vector named **pair** to hold the results of each iteration.

```

proc iml;
n=23;
numberIterations=1000;
call randseed(802);
pair = j(numberIterations,1,.);
do iteration=1 to numberIterations;

```

- b. Draw 23 birthdates using the SAMPLE function. (Dates can be represented as the numbers 1 through 365.)

```
dates = 1:365;
birthDates=sample(dates,n);
```

- c. Check whether any two birthdates are the same. (Hint: Use the UNIQUE function.)

```
uniqueDates=unique(birthDates);
```

- d. If at least two birthdates are the same, set the variable **pair** to 1. Otherwise, set **pair** to zero.

```
if ncol(uniqueDates) < n then pair[iteration]=1;
else pair[iteration]=0;
end;
```

- e. Calculate the proportion of iterations in which a pair was found.

```
proportion=pair[:];
print proportion;
quit;
```

proportion

0.506

- f. How can you avoid using the DO loop for this simulation?

Simulate dates in a matrix with dimension (number of iterations) by (number of people).

```
proc iml;
n=23;
numberIterations=1000;
call randseed(23571113);
prob=j(364,1,1/365);
birthDates=j(numberIterations,n,.);
call randgen(birthDates,"Table",prob);
```

After you enter IML and set the random number seed, this version of the birthday problem simulation creates a vector of probabilities, **prob**, to be used as a parameter for the table distribution. The program then creates a 1000 x 23 matrix of missing values and assigns it to **birthDates**. The program then fills the **birthDates** matrix with values drawn from the table distribution with parameter **prob**.

```
rowUnique=countunique(birthDates,"ROW");
proportion=(rowUnique < n) [+] / numberIterations;
print proportion;
quit;
```

Specifying the “ROW” option for **countunique** tells the function to calculate the number of unique elements in each row of the argument matrix. Then **rowUnique** is assigned the number of unique birthdates for each row in **birthDates**. The syntax **(rowUnique < n) [+]** counts the number of rows that contain fewer than 23 unique birthdates. Dividing the number of unique birthdates by the number of iterations provides the proportion of samples containing at least two matching birthdates.

7. Using IML to Identify the Sampling Distribution for a Uniform Random Variable

- a. Invoke PROC IML and create a function with inputs, n (for sample size), and a minimum and maximum value. Within the user-defined function create a vector with n random numbers from the distribution $\text{Uniform}(\min, \max)$.

```
proc iml;
  start mySample(n,min,max);
    x = randfun(n,"Uniform",min,max);
    return(x);
  finish;
```

- b. Set the sample size to 30, the minimum to 5, and the maximum to 10. Set the number of replications to 1000 and the seed to 802. Lastly, initialize a vector to store the mean of each simulation value.

```
n = 30;
min = 5;
max = 10;
call randseed(802);
reps = 1000;
vec = j(reps,1,);
```

- c. Begin a DO loop and use the user-defined function to compute and store the mean of each sample for each iteration.

```
do i=1 to reps;
  vec[i] = mean(mySample(n,min,max));
end;
```

- d. Pass the simulation results to a SAS data set.

```
create sp4r.simulation var {vec};
append;
close sp4r.simulation;
```

- e. Submit PROC UNIVARIATE code to compute the BasicMeasures table and a histogram of the simulation results.

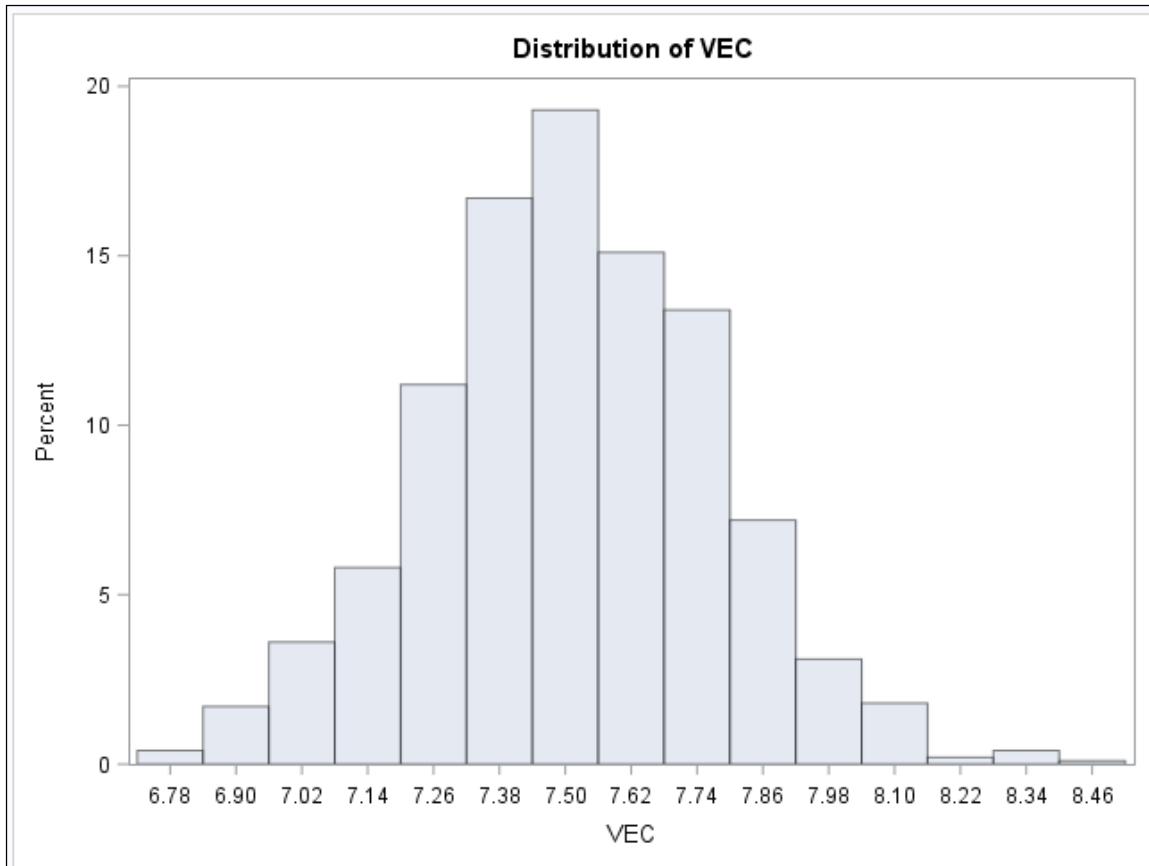
```
submit;
  ods select basicmeasures histogram;
  proc univariate data=sp4r.simulation;
    var vec;
    histogram vec;
  run;
endsubmit;
quit;
```

IML Output

Basic Statistical Measures

	Location	Variability	
Mean	7.509750	Std Deviation	0.26461
Median	7.512882	Variance	0.07002
Mode	.	Range	1.72576
		Interquartile Range	0.35543

Mean	7.509750	Std Deviation	0.26461
Median	7.512882	Variance	0.07002
Mode	.	Range	1.72576



8. Redoing the Previous Exercise Using a BY Statement and Avoiding the DO Loop

- a. Invoke PROC IML and create a variable corresponding to 1,000 simulations, a sample size of 30, a minimum value of 5, and a maximum value of 10. Set a seed of 802.

```
proc iml;
  simulations = 1000;
  sampleSize = 30;
  min = 5;
  max = 10;
  call randseed(802)
```

- b. Create a vector that represents the simulation number for each sampled data point. Let each simulation number repeat 30 times. (Hint: Use the REPEAT function.)

```
simulationNumber = 1:simulations;
each = j(simulations,1,sampleSize);
simulationNumber = repeat(simulationNumber,each) `;
```

- c. Generate data from a uniform random variable with minimum and maximum values of 5 and 10 respectively. Notice that the total number of observations should be the number of simulations times the sample size ($1,000 \times 30 = 30,000$).

```
total = simulations*sampleSize;
vec = randfun(total,"Uniform",min,max);
```

- d. Pass the vector of random numbers and simulation number to a SAS data set.

```
create sp4r.simulation var {simulationNumber vec};
append;
close sp4r.simulation;
```

- e. Use an appropriate procedure to output the mean of each sample from each simulation using a BY statement. Be sure to use the ODS SELECT NONE and ODS SELECT DEFAULT statements.

```
submit;
ods select none;
proc means data=sp4r.simulation;
by simulationNumber;
var vec;
output out=sp4r.out mean=mean;
run;
ods select default;
endsubmit;
```

- f. Read the means back into IML.

```
use sp4r.out;
read all var {mean} into means;
close sp4r.out;
```

- g. Compute and print the mean, standard deviation, 2.5% percentile, and the 97.5% percentile for the means data.

```
mean = mean(means);
sd = std(means);
call qntl(percentiles,means,.025, .975);

out = mean//sd//percentiles;
reset noname;
print out[colname="Mu"
           rowname={"Mean","Standard Deviation","LCL","UCL"}];
quit;
```

IML Output

Mu	
Mean	7.5097502
Standard Deviation	0.2646051
LCL	6.9767451
UCL	8.0408154

End of Solutions

Solutions to Student Activities (Polls/Quizzes)

7.01 Multiple Answer Poll – Correct Answers

Suppose you want to print your salary for the week. Assume that you worked 40 hours and earn \$9.35 per hour. Which of the following show the correct syntax for printing your salary? (Select all that apply.)

You do not need brackets to assign a scalar.

- a. $\text{Y}=40*9.35;$ Print Y;
- b. $40*9.35;$
- c. $\text{Print}(40*9.35);$
- d. $\text{Print } 40*9.35;$

11

Correct Answers: A, C

B cannot be correct because SAS does not have a command line.

D is not correct because there are no parentheses.

When multiplying scalars, you can use either the * or the # operator symbols.

7.02 Multiple Choice Poll – Correct Answer

Let X be an m -by- n matrix. How would you use a SAS reduction operator to reproduce the **rowMeans()** and **min()** functions in R?

- a. $\text{X}[:,:]$ and $\text{X}[><,]$
- b. $\text{X}[:,:]$ and $\text{X}(>:<,>:<)$
- c. $\text{X}\{:,:\}$ and $\text{X}[><,><]$
- d. $\text{X}[:,:]$ and $\text{X}[><,><]$

19

Correct Answer: D

To take the minimum of the entire matrix, use the appropriate reduction operator in both the row and column argument.

7.03 Poll – Correct Answer

The PROC IML code below prints the 0.75 quantile from matrix **X**.

- True
- False

```
Q = call qntl(X,{0.75});
print Q;
```

31

Correct Answer: False

Subroutines do not use an assignment statement.

7.04 Quiz – Correct Answer

Navigate to the SAS/IML documentation and peruse the statements, functions, and subroutines. Choose a few that look familiar to you and see what they do.

Next, find the LOC function and see what it does.

$$X = \begin{bmatrix} 4 & 2 & -1 \\ 3 & -1 & -4 \\ -9 & 5 & 9 \end{bmatrix}$$

Use the LOC function to change the elements according to a specified criterion.

The screenshot shows the SAS/IML interface with a code editor window. A blue arrow points from the highlighted line of code to the resulting matrix X . The code is:

```
x[loc(x<0)]=0;
```

The resulting matrix X is:

$$X = \begin{bmatrix} 4 & 2 & 0 \\ 3 & 0 & 0 \\ 0 & 5 & 9 \end{bmatrix}$$

47

The LOC function returns a row vector containing indices of the elements in a matrix that satisfy a criterion. If an expression is not specified, the LOC function finds elements that are nonzero and nonmissing.

7.05 Multiple Answer Poll – Correct Answers

Which of the following statements about SAS modules are true?

- a. Modules are defined by START and FINISH keywords.
- b. Functions use the RETURN statement.
- c. The RETURN statement can handle multiple arguments.
- d. Subroutines can be executed by the CALL statement.

57

Correct Answer: A, B, D

7.06 Multiple Choice Poll – Correct Answer

How do you recall the module **rock** and matrix **pony** into a new SAS/IML session from the **mycat** catalog in the **Work** directory?

- a. RESET STORAGE; LOAD module=(rock) pony;
- b. RESET STORAGE=mycat; LOAD module=(rock) pony;
- c. STORAGE=mycat; LOAD module=(rock) pony;
- d. RESET STORAGE=mycat; LOAD rock pony;

63

Correct Answer: B

7.07 Multiple Answer Poll – Correct Answers

Which statements are true regarding importing SAS data sets and exporting IML matrices? (Select all that apply.)

- a. The statements USE, READ, and CLOSE are used to pass a SAS data set into IML.
- b. The statements CREATE, APPEND, and CLOSE are used to pass an IML matrix to a SAS data set.
- c. Names of IML vectors are passed to the SAS data set as variable names.
- d. The user must specify the column names when creating a SAS data set from an IML matrix.

80

Correct Answer: A, B, C

Chapter 8 A Bridge between SAS and R

8.1 Calling R from IML	8-3
Demonstration: Bootstrap with R from IML	8-12
Exercises	8-17
8.2 Calling R from Base SAS Java API (Self-Study)	8-18
8.3 Calling R from SAS Enterprise Miner (Self-Study)	8-26
Demonstration: Getting Started with Enterprise Miner and the Open Source Integration Node	8-37
8.4 Solutions	8-54
Solutions to Exercises	8-54
Solutions to Student Activities (Polls/Quizzes)	8-56

8.1 Calling R from IML

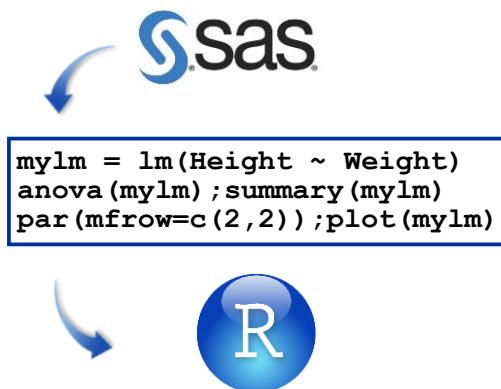
Objectives

- Enable R language statements to connect SAS and R.
- Transfer SAS data to R.
- Submit R code in an IML script.
- Transfer R results to SAS.

3

Motivation

Freely write R code within a SAS script, send it to the open source software, and retrieve the results.



4

This chapter shows how to incorporate an R script into three different SAS environments.

Open Source Integration

SAS views open source software as a complementary resource. You can easily assimilate open source tools into a SAS script to enjoy the following benefits:

- supplement SAS capabilities
- implement new methodology quickly
- compare methods
- create diverse plots

5

Open Source Integration

Integrate open source software into three different SAS environments.

- Base SAS
 - Execute open source code via a DATA step.
- Interactive matrix language (IML)
 - Execute open source code with the SUBMIT statement.
- SAS Enterprise Miner
 - Execute open source code via the Open Source Integration node.

6

This section focuses on using R in the SAS/IML session.



To run R with SAS, R must be installed on the same machine as SAS. Because SAS University Edition installs on a virtual machine where R cannot be installed, R cannot be used with SAS University Edition.

R Language Statements

The RLANG system option must be enabled in order to call R from SAS. To view your setting, run the following syntax:

```
proc options option=rlang;
run;
```

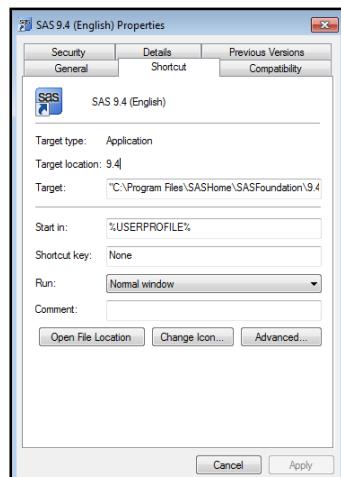
One of these two results is printed to the log:

- NORLANG – You do not have permission to call R from SAS.
- RLANG – You have permission to call R from SAS.

7

Enabling R Language Statements

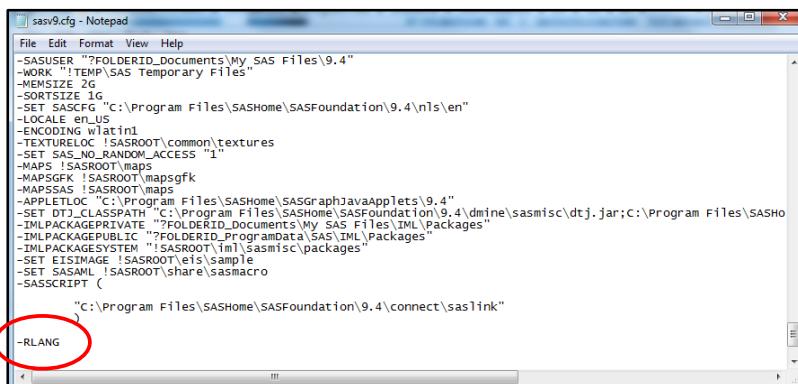
- To find the SASV9.CFG file, right-click the SAS icon and click **Properties**.
- The **Target** field names the location of the SASV9.CFG file:
-CONFIG "C:\Program Files\SASHome\SASFoundation\9.4\nls\en\sasv9.cfg"



8

Enabling R Language Statements

Navigate to the **SASV9.CFG** file and add the **-RLANG** option. Be sure to save the changes.



```
sasv9.cfg - Notepad
File Edit Format View Help
-SASUSER "FOLDERID_Documents\My SAS Files\9.4"
-WORK "FOLDERID_SAS Temporary Files"
-MEMSIZE 2G
-SORTSIZE 2G
-SET SASCFG "C:\Program Files\SASHOME\SASFoundation\9.4\nls\en"
-LOCATE en_US
-ENCODING UTF8
-TEXTURELOC !SASROOT\common\textures
-SET SAS_NO_RANDOM_ACCESS "1"
-MAPS !SASROOT\maps
-MAPSGFK !SASROOT\maps\gfk
-MAPSSAS !SASROOT\maps
-APPENDOC "C:\Program Files\SASHOME\SASGraphJavaApplets\9.4"
-SET DTJ_CLASSPATH "C:\Program Files\SASHOME\SASFoundation\9.4\dmne\sasmisc\dtj.jar;C:\Program Files\SASHOME\ProgramData\SAS\IML\Packages"
-IML PACKAGEPRIVATE "?FOLDERID_Documents\My SAS Files\IML\Packages"
-IML PACKAGEPUBLIC "?FOLDERID_ProgramData\SAS\IML\Packages"
-IML PACKAGESYSTEM "?SASROOT\iml\sasmisc\packages"
-SET EISIMAGE !SASROOT\els\sample
-SET SASML !SASROOT\share\sasmacro
-SASSCRIPT (
    "C:\Program Files\SASHOME\SASFoundation\9.4\connect\saslink"
)
-RLANG
```

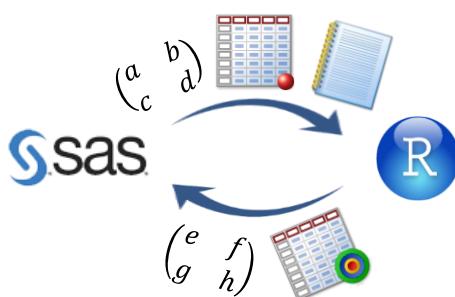
9

The **RLANG** option can be changed only at start-up because you are altering the SAS configuration file. Thus, close your SAS session if it is open and proceed to enable R language statements.

Working with R in IML

After enabling R language statements, use IML to do the following:

- send IML matrices and SAS data sets to R
- submit R code in the IML script
- return R results as IML matrices or SAS data sets



10

8.01 Multiple Answer Poll

Choose the correct statements.

- a. R can be called from Base SAS, SAS/IML, and SAS Enterprise Miner.
- b. -RLANG must be added to the SAS configuration file.
- c. PROC OPTIONS is used to test the SAS and R connection.
- d. You should leave SAS open when altering the configuration file.

11

Exporting SAS Data Sets

General form of the EXPORTDATASETTOR call:

```
CALL EXPORTDATASETTOR("SAS-data-set", "R-data-frame");
```

- The first argument to the subroutine is the name of the SAS data set to be exported.
- The second argument to the subroutine is the name of the R data frame to be created.
- Names of SAS variables are transferred to the R data frame.

13

Exporting IML Matrices

General form of the EXPORTMATRIXTOR call:

```
CALL EXPORTMATRIXTOR(IML-matrix, "R-matrix");
```

- The first argument to the subroutine is the name of the IML matrix to be exported.
- The second argument to the subroutine is the name of the R matrix to be created.
- Names of matrix columns are not transferred to the R matrix.

14

Submitting R Syntax

After exporting data to R, submit R code with the SUBMIT statement and the R option.

- Adding the R option to the SUBMIT statement sends syntax in the SUBMIT block to R instead of SAS.

```
imlMatrix = {0 1, 1 2, 3 5, 8 13};
call ExportMatrixToR(imlMatrix,"rmatrix");
submit / R;
  print(rmatrix)
endsubmit;
```

```
SUBMIT / R;
  R statements
ENDSUBMIT;
```

15

Viewing the Output

All R command line output is displayed in the Results Viewer.

- The output is printed in R format.

```
A1 A2
[1,] 0 1
[2,] 1 2
[3,] 3 5
[4,] 8 13
```

16

Importing R Objects into IML Matrices

General form of the IMPORTMATRIXFROMR call:

```
CALL IMPORTMATRIXFROMR(IML-matrix, "R-object");
```

- The first argument to the subroutine is the name of the IML matrix to be created.
- The second argument to the subroutine is the name of the R object to be imported.
- If the R object is a mix of character and numeric variables, the type of the new IML matrix matches the type of the first variable in the R object.

17

Example: SAS to R and Back

- Export an IML matrix to R.
- Submit R code.
- Return the results.

```
imlMatrix = {0 1, 1 2, 3 5, 8 13};
call ExportMatrixToR(imlMatrix,"rmatrix");
submit / R;
  rmatrix = rmatrix + 49
endsubmit;
call ImportMatrixFromR(NewMatrix,"rmatrix");
print NewMatrix;
```

NewMatrix	
49	50
50	51
52	54
57	62

18

Importing R Objects into SAS Data Sets

General form of the IMPORTDATASETFROMR call:

```
CALL IMPORTDATASETFROMR("SAS-data-set", "R-object");
```

- The first argument to the subroutine is the name of the SAS data set to be created.
- The second argument to the subroutine is the name of the R object to be imported.
- The IMPORTDATASETFROMR call can transfer a mix of character and numeric variables.

19

8.02 Poll

The code below prints the first column of the data frame in the SAS Results Viewer.

- True
- False

```
call ExportDataSetToR("dog","rmatrix");
submit;
  rmatrix[,1]
endsubmit;
```



Bootstrap with R from IML

SP4R08d01.sas

The **fish** data set contains measurements of 159 fish that were caught in Finland's Lake Laengelmavesi. This script produces bootstrap estimates of the upper and lower 95% confidence interval limits for the mean weight of the fish.

1. Use PROC UNIVARIATE to analyze the **Weight** variable of the **fish** data set. Request only the BasicMeasures table and a histogram.

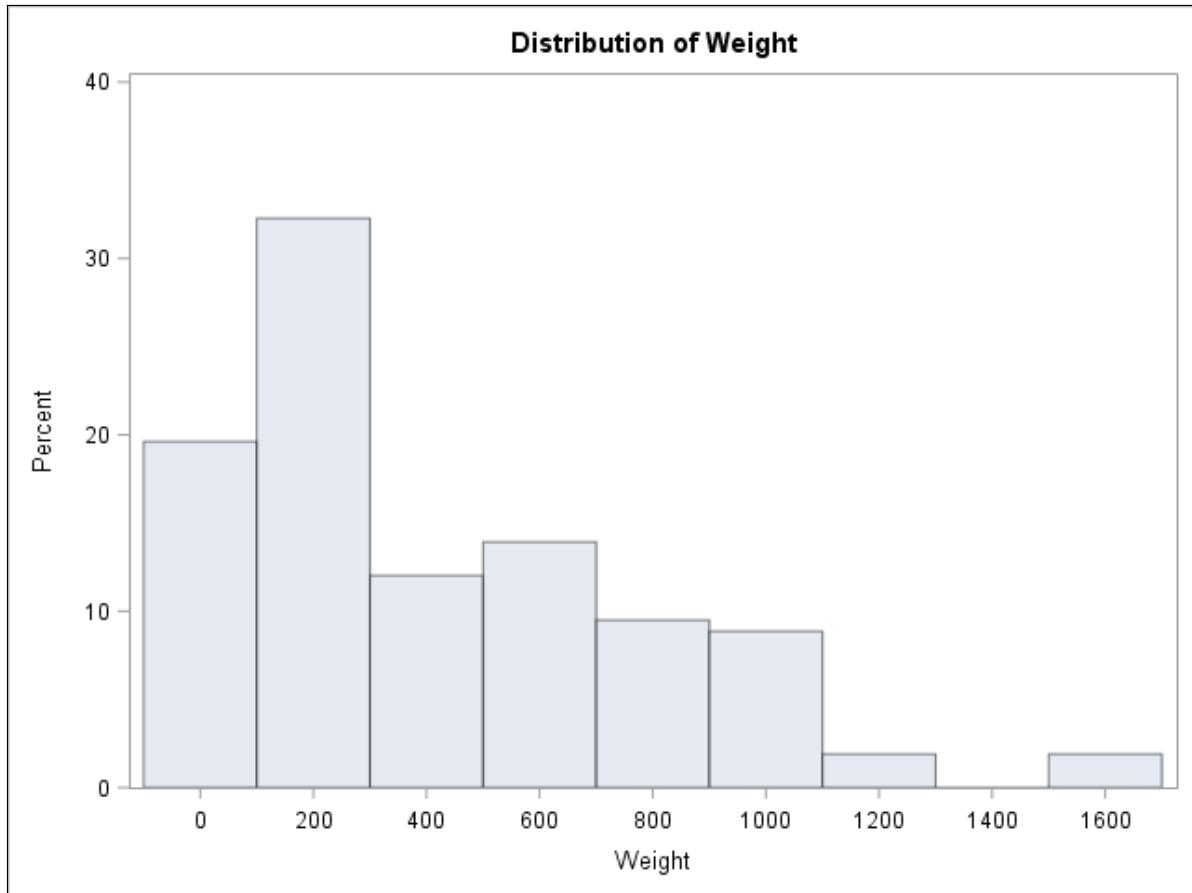
```
ods select basicmeasures histogram;
proc univariate data=sp4r.fish;
  var Weight;
  histogram Weight;
run;
```

The UNIVARIATE Procedure
Variable: Weight

Basic Statistical Measures

Location		Variability	
Mean	398.6956	Std Deviation	359.08620
Median	272.5000	Variance	128943
Mode	300.0000	Range	1650
		Interquartile Range	530.00000

Location		Variability	
Mean	398.6956	Std Deviation	359.08620
Median	272.5000	Variance	128943
Mode	300.0000	Range	1650
		Interquartile Range	530.00000



- To conduct the bootstrap simulation, use the **Boot** package in R. Use the **install.packages** function if the **Boot** package was not downloaded.

```
proc iml;
  submit / r;
    install.packages ("boot")
  endsubmit;
quit;
```

- Use the EXPORTDATASETTOR subroutine to send the **fish** data set to an R data frame. Recall that sending a SAS data set to R passes the variable names to the R data frame.

```
proc iml;
  call ExportDataSetToR("sp4r.fish","fish");
```

- Load the **Boot** package with the **library()** function. The **boot** function in R requires the user to generate a separate function that produces the estimates to be bootstrapped. In this case, the **bootMean** function is created to calculate the mean for a specified data frame and variable. The **boot** function performs bootstrapping for the user-defined function, data frame, and number of replicates. The results are stored in Results.

```
submit / r;
  library(boot)
  set.seed(802)
  numreps = 1000
```

```

bootMean <- function(data,variable,index) {
  attach(data)
  result <- mean(variable[index],na.rm=TRUE)
  detach(data)
  return(result)
}

results <- boot(data=fish,statistic=bootMean,
  R=numreps,variable=Weight)

```

5. The **boot.ci()** function generates confidence intervals for a bootstrapped parameter. The **type="perc"** option requests only percentiles according to the **conf=** specification. The **index=** option requests that confidence intervals be generated for the first parameter (the mean).

```

boot.ci(results, conf=0.95, type="perc", index=1)
plot(results)

```

Both the **boot.ci()** and **plot()** functions produce output as follows:

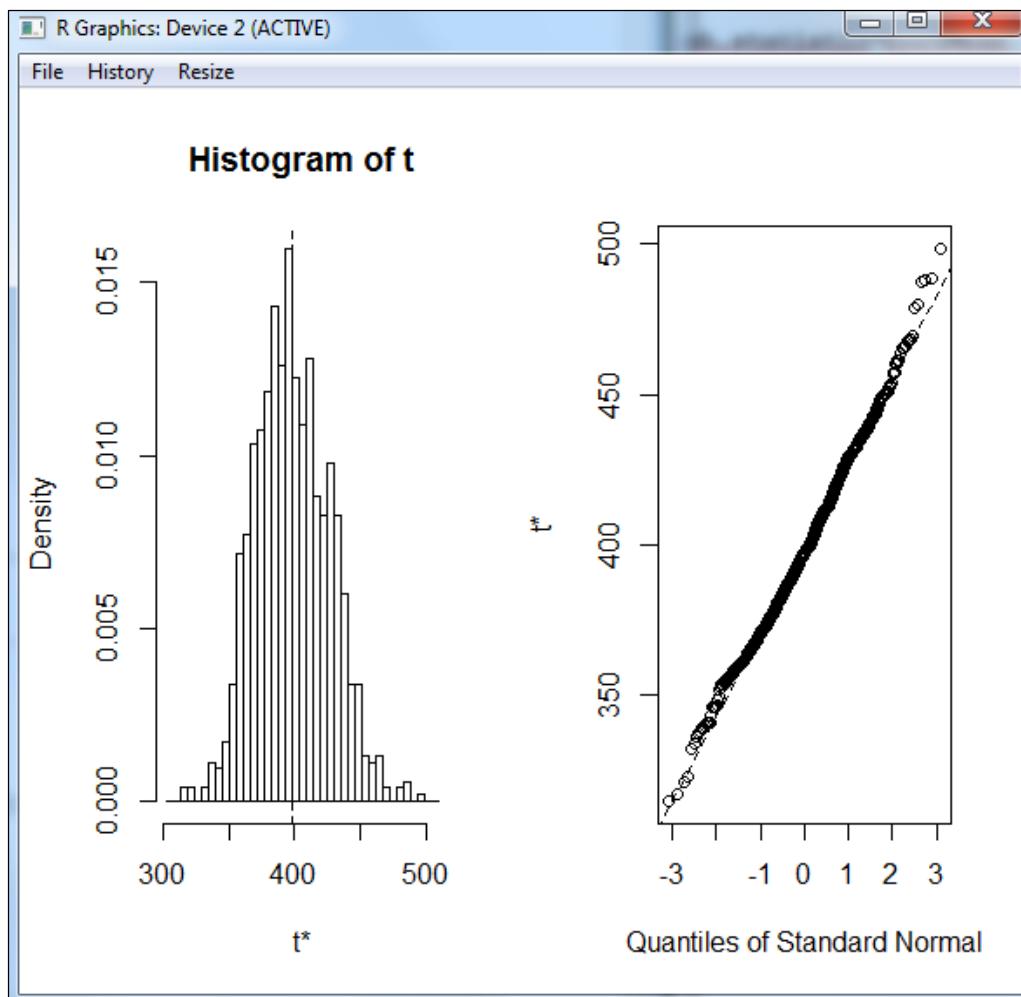
```

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = results, conf = 0.95, type = "perc", index = 1)

Intervals :
Level      Percentile
95%    (348.7, 453.5 )
Calculations and Intervals on Original Scale

```



6. Notice that the R command line output is printed in the SAS window and the **plot()** function output is plotted in the R graphics window. Save the plot by selecting **File** \Rightarrow **Save As** in the plot window.

```
boots <- data.frame("boots"=results$t)
endsubmit;
```

The bootstrap estimates are saved in an R data frame with the variable name **Boots**.

The R graphics are produced only in the SAS windowing environment.

7. The IMPORTDATASETFROMR subroutine is used to call the R data frame bootstrap estimates to a SAS data set. The first ten observations are printed using PROC PRINT. Notice that the R data frame variable name is passed to the SAS data set.

```
call ImportDataSetFromR("sp4r.RData","boots");

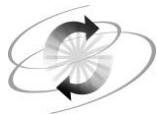
submit;
proc print data=RData (obs=10);
run;
endsubmit;
```

Obs	boots
1	409.525
2	393.014
3	397.510
4	385.338
5	397.020
6	371.296
7	365.806
8	417.251
9	352.568
10	443.639

```
quit;
```

8. End the IML session with the QUIT statement. Notice that the R graphics window closes when the QUIT statements is submitted.

End of Demonstration



Exercises

1. Comparing Multiple Regression Estimates in SAS and R

- a. Begin by invoking PROC IML and exporting the **fish** data set to R as a data frame with the name **Fish**.
- b. Fit a linear model with **Weight** as the dependent variable and **Height** and **Width** as the independent variables using the **Im()** function. Store the object and use the **summary()** function to print model estimates.
- c. Import the parameter estimates into an IML matrix. Recall that the parameter estimates are stored under the name **Coefficients** in the R object.
- d. Run the same analysis in SAS using PROC REG. Output the parameter estimates using the OUTTEST= option in the PROC REG statement.
- e. Import the SAS coefficients into IML using the USE, READ, and CLOSE statements.
- f. Print the SAS coefficients and R coefficients side by side along with the difference between the estimates.

SAS_COEFFICIENTS	R_COEFFICIENTS	DIFFERENCE
-433.6525	-433.6525	-5.12E-13
5.5068475	5.5068475	-7.66E-13
177.44357	177.44357	1.648E-12

End of Exercises

8.2 Calling R from Base SAS Java API (Self-Study)

Objectives

- Create the Java pipeline from SAS to R.
- Use a DATA step to run an R script.

25

Motivation

Write and submit R code inside a DATA step.



```
data _null_;  
...  
r_script = "mylm=lm(Height ~ Weight);  
anova(mylm);summary(mylm);"  
...  
run;
```



26

JAVA Pipeline

The SAS DATA step does not pass an R script directly from SAS to R.

- JAVA must be used as an intermediate tool.



- This path is not inherent. You must manually create the connection from SAS to Java to R.

27

Setup

1. Download and extract the project ZIP file SAS_Base_OpenSrcIntegration.zip from <https://communities.sas.com/docs/DOC-10746>.

Tip: Open Source Integration Using the Base SAS Java Object

Version 1.1
created by Pritchett on Mar 21, 2015 3:17 PM, last modified by Pritchett on Apr 22, 2015 4:06 PM

This tip introduces a simple and effective method that uses Base SAS to interact with other tools like R and Python. Running scripts from R, Python, and other languages within SAS will enable you to create hybrid data science and machine learning solutions.

Here's a sneak peak code snippet:

```

data _null_;
  python_rtn_val $;
  *** Python program takes working directory as first argument;
  python_pgm = "D:\HOME\DIR\digitizedata_rvm.py";
  python_arg1 = "D:\HOME\DIR";
  python_call = get('!', trim(python_pgm, ' '), trim(python_arg1, ''));
  declare JavaExec J("java.SASJavaExec", "SASJavaExec", python_call);
  J->callInMethod("executeProcess", rtn_val);
run;
  
```

In this code snippet, the path to a Python modeling script is assigned to the variable `python_pgm`; and the path specifying a directory containing sample data is assigned to the variable `python_arg1`. `python_pgm` and `python_arg1` are concatenated into the variable `python_call`. `python_call` along with the path to a python executable are passed to the Java class `SASJavaExec` using the Base SAS Java Object. `SASJavaExec` runs the Python script with its command line arguments and passes any output or errors back to the SAS log.

You can download [SAS Java Exec](#), a detailed white paper that explains the new method, and other example materials attached to this post. You will need suitable versions of R, Python, and Java installed alongside SAS to run the attached example code. This open source code is also available on [GitHub](#): [erighen/SAS_Base_OpenSrcIntegration at master · sassoftware/erighen/integration · GitHub](#).

The method works inside SAS Enterprise Miner too. The tip [Tip: How to execute a Python script in SAS Enterprise Miner](#) explains how to deploy `SASJavaExec` with Enterprise Miner.

28

The download prompts you to save the files on your computer at C:\SGF2015\OpenSrcIntegration. The subsequent steps assume that this is the location of the Java files.

Setup

2. Download the Java Development Kit (JDK) from oracle.com.

www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

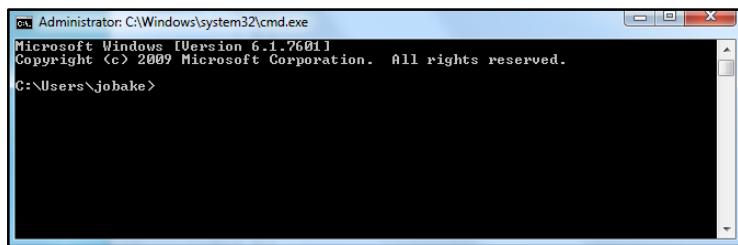
Java SE Development Kit 7u79
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Product / File Description	File Size	Download
Linux x86	130.4 MB	jdk-7u79-linux-i586.rpm
Linux x86	147.6 MB	jdk-7u79-linux-i586.tar.gz
Linux x64	131.69 MB	jdk-7u79-linux-x64.rpm
Linux x64	148.4 MB	jdk-7u79-linux-x64.tar.gz
Mac OS X x64	196.89 MB	jdk-7u79-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.79 MB	jdk-7u79-solaris-i586.tar.Z
Solaris x86	96.66 MB	jdk-7u79-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.67 MB	jdk-7u79-solaris-x64.tar.Z
Solaris x64	16.38 MB	jdk-7u79-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	140 MB	jdk-7u79-solaris-sparc.tar.Z
Solaris SPARC	99.4 MB	jdk-7u79-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	24 MB	jdk-7u79-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.4 MB	jdk-7u79-solaris-sparcv9.tar.gz
Windows x86	138.31 MB	jdk-7u79-windows-i586.exe
Windows x64	140.06 MB	jdk-7u79-windows-x64.exe

29

Setup

3. Compile the Java classes.
- Open the Windows command line.



- Enter the following:
 - **cd C:\SGF2015\OpenSrcIntegration**
 - **"C:\Program Files\Java\jdk1.7.0_25\bin\javac"**
 - src/dev/* -d bin**

30

Downloading the Java Development Kit gives you access to the JAVAC command on the Windows command line. The JAVAC command is used to compile the extracted Java files, and creates the connection from SAS to R.

Setup

4. Add the following location of the compiled Java classes to the SAS configuration file.

C:\Program Files\SASHome\SASFoundation\9.4\nls\en\sasv9.cfg

```
sasv9.cfg - Notepad
File Edit Format View Help
-SET SAS_NO_RANDOM_ACCESS "1"
-SET !SASROOT\applets
-MAPGFX !SASROOT\maps\gfx
-MAPSSAS !SASROOT\maps
-APPLETLOC "C:\Program Files\SASHome\SASGraphJavaApplets\9.4"
-SET DTJ_CLASSPATH "C:\Program Files\SASHome\SASFoundation\9.4\dmine\sasmisc\dtj
-IML PACKAGEPRIVATE "?FOLDERID_Documents\My SAS Files\IML\Packages"
-IML PACKAGEPUBLIC "FOLDERID_Programdata\SAS\IML\Packages"
-IML PACKAGETESTED "FOLDERID_Programdata\SAS\IML\Packages"
-SET EISIMAGE !SASROOT\ejis\sample
-SET SASAM! !SASROOT\share\sasmacro
-SASSCRIPT (
)
-C:\Program Files\SASHome\SASFoundation\9.4\connect\saslink"
-RLANG
-SET CLASSPATH "C:\SGF2015\OpenSrcIntegration\bin"
```

- -SET CLASSPATH "C:\SGF2015\OpenSrcIntegration\bin"

31

This is the same SAS configuration file that is used to add the RLANG option from the previous section.

Setup

5. Ensure that the Java classes are compiled and that the CLASSPATH is set correctly.

- Set a working directory and the Java directory.

```
%let WORK_DIR = C:\SGF2015\OpenSrcIntegration;
%let JAVA_BIN_DIR = &WORK_DIR.\bin;
```

- Validate the Java pipeline.

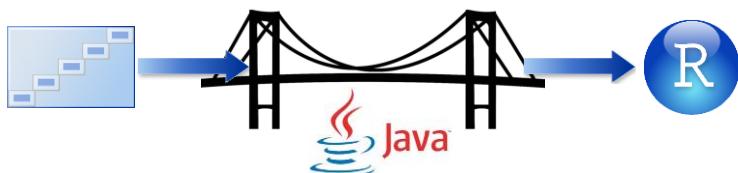
```
data _null_;
length _x1 $ 32767;
_x1 = sysget('CLASSPATH');
_x2 = index(upcase(trim(_x1)),
%upcase("&JAVA_BIN_DIR"));
if _x2 = 0 then put "ERROR: Invalid Java
Classpath.";
run;
```

32

If the Java pipeline is created correctly, the SAS log is empty. Otherwise, the log contains “ERROR: Invalid Java Classpath.”.

Setup Complete!

- The Java pipeline is built.
- You are now ready to submit R code inside a DATA step.



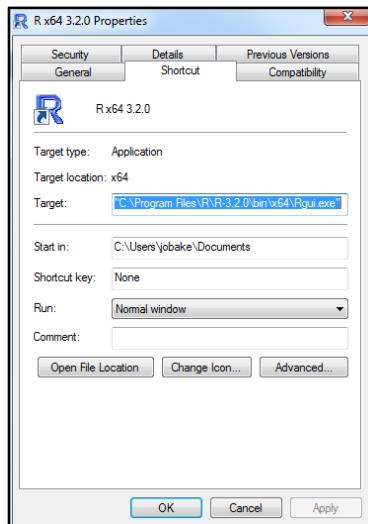
33

R Command Line

Set the R system location.

- Right-click the **R desktop** icon and select **Properties**.
- Copy the value from the **Target** field into SAS and create a macro variable.
- Replace the **Rgui** text with **Rscript**.

```
%let R_EXEC_COMMAND =
C:\Program Files\R\R-3.2.0\bin\x64\
Rscript.exe;
```



34

Changing the text to **Rscript** sets the path to the R command line.

This location tells Java where to pass the DATA step R script.

DATA Step Syntax

The DATA step below is used to submit R code.

- Add your R script to the **R SCRIPT** field.

```
data _NULL_;
length rtn_val 8;
length r_call $ 32000;

r_expr = "-e";
r_script = "R_SCRIPT";

r_call = catt('', r_expr, ' ', r_script, '');
declare javaobj j("dev.SASJavaExec", "&R_EXEC_COMMAND",
                  r_call);
j.callIntMethod("executeProcess", rtn_val);

run;
```

35

Selected DATA step statements and options:

NULL	tells SAS not to make a data table but run the DATA step.
length rtn_val	initializes the rtn_val variable to return R log output.
length r_call	initializes the length of the R script to be created.
r_expr="-e"	sets this variable to <code>-e</code> and tells Java that the DATA step is an expression.
r_call	concatenates the r_expr and r_script text strings. The r_call text string is the string that is passed from the DATA step to Java and finally R.
declare javaobj	creates the Java object pipeline. The second argument is the R command line macro variable and the third argument is the R script.
j.callIntMethod	returns the R command line output.

Caution

The DATA step method of calling R is unable to do either of the following tasks:

- send a SAS data set to R
 - Use the EXPORT procedure to save data outside of the SAS environment. Read in the data file using an R statement.
 - return a matrix or data frame to a SAS data set
 - Save the matrix or data frame in R and use a PROC IMPORT statement to create a SAS data set.
-  R command line output is returned to the SAS log in SAS format.

36

Writing the R Script

The R script must

- end each R statement with a semicolon
- use single quotation marks only.
 - The double quotation marks are used to begin and end the R script.

```
data _NULL_;
...
r_script = "library(fields);
setwd('C:/SGF2015/OpenSrcIntegration');
locations = read.csv('locations.csv');
dist_mat = rdist(locations);
write.table(dist_mat,'dist_mat.csv',
            sep = ',',row.names=F);";
...
run;
```

Export data to CSV.

37

The R script is condensed by removing all trailing blanks and is then concatenated with the **-e** variable. (This is done using the CATT function.) Thus, the R script sent to Java is as follows:

```
"-e library(fields);setwd('C:/SGF2015/OpenSrcIntegration');..."
```

The only blank is between the **-e** and the rest of the script. This is why it is necessary to use a semicolon after each statement. The R script is passed to R on a single line.

The script above begins by unpacking the fields R package. The **read.csv()** function reads in the **locations** data set in the directory specified in the **setwd()** function. PROC EXPORT can be used to export a SAS data set to a CSV and the directory can be chosen. The **rdist()** function creates a results matrix of the Euclidean distances between each location pair. Finally, the **write.table()** function exports the R results back to the working directory set by the **setwd()** function. Thus, the DATA step runs properly if the results from R are stored in the desired directory. The results can then be returned to SAS using PROC IMPORT.

Calling the R Script

Alternatively, call and run a saved R script.

- This DATA step runs the dist.R file.

```
data _null_;
length rtn_val 8;
r_pgm = "&WORK_DIR.\dist.R";
r_arg1 = "&WORK_DIR";
r_call = cat('`', trim(r_pgm), ' `',
            trim(r_arg1), '`');
...
run;
```

38

Add the DECLARE JAVAOBJ and J.CALLINTMETHOD statements to the end of the DATA step.

The rdist.R file is simply a saved R file. It does not require quoting the R code or using semicolons. The code from the previous slide would be saved as follows:

```
library(fields)
setwd('C:/SGF2015/OpenSrcIntegration')
locations = read.csv('locations.csv')
dist_mat = rdist(locations)
write.table(dist_mat,'dist_mat.csv',sep = ',',row.names=F)
```

8.3 Calling R from SAS Enterprise Miner (Self-Study)

Objectives

- Navigate the SAS Enterprise Miner interface.
- Create a simple process flow.
- Use the Open Source Integration node.

40

Motivation

Freely write R code within SAS Enterprise Miner's Open Source Integration node.  Send it to R, and retrieve the results.



```
&EMR_MODEL <- lm(rHeight ~ Weight)
```



41

SAS Enterprise Miner

The SAS Enterprise Miner interface streamlines and simplifies common tasks associated with applied analysis.

Process Flow:

```

graph LR
    data[data] --> InputData[Input Data]
    InputData --> DataPartition[Data Partition]
    DataPartition --> Regression[Regression]
    DataPartition --> DecisionTree[Decision Tree]
    Regression --> ModelComparison[Model Comparison]
    DecisionTree --> ModelComparison
  
```

42

The process flow indicates each step in the analysis from data entry to modeling the data.

SAS Enterprise Miner

```

graph LR
    PVA7NK[PVA7NK] --> Replacement[Replacement]
    Replacement --> DataPartition[Data Partition]
    DataPartition --> Impute[Impute]
    Impute --> Regression[Regression]
    DataPartition --> DecisionTree[Decision Tree]
  
```

43

The SAS Enterprise Miner interface is a point-and-click interface. It offers secure analysis management and provides a wide variety of tools with a consistent graphical interface. You can customize it by incorporating your choice of analysis methods and tools.

SAS Enterprise Miner: Interface Tour

The screenshot shows the SAS Enterprise Miner interface with a red box highlighting the 'My Project' sidebar. The 'ORGANICS' project is selected. The main panel displays a flow diagram titled 'Project Panel' for 'Predictive Analysis'. The diagram consists of nodes: 'PVAS7NK' → 'All Replacement' → 'Data Partition' → 'Impute' → 'Regression'. A 'Decision Tree' node is also connected to the 'Impute' node.

44

The *project panel* manages and views data sources, diagrams, results, and project users.

SAS Enterprise Miner: Interface Tour

The screenshot shows the SAS Enterprise Miner interface with a red box highlighting the properties table in the 'Properties Panel'. The table has sections for 'General', 'Train', and 'Regression Type'. Under 'General', 'Node ID' is set to 'Reg'. Under 'Train', 'Regression Type' is set to 'Logistic Regression' with 'Function' set to 'Logit'. Other settings like 'Main Effects' and 'Two-Factor Interaction' are also listed.

45

The *properties panel* enables you to view and edit the settings of data sources, diagrams, nodes, results, and users.

SAS Enterprise Miner: Interface Tour

The screenshot shows the SAS Enterprise Miner interface. On the left is the Project Explorer with nodes like 'ORGANICS' and 'PVA97NK'. In the center is the 'Predictive Analysis' workspace showing a process flow: PVA97NK → Fill Replacement → Data Partition → Deploy → Regression. A 'Help Panel' window is open at the bottom, displaying the properties for the selected node 'PVA97NK'. The 'General' tab is selected, showing fields such as 'Node ID' (Reg), 'Main Effects' (Yes), and 'Two-Factor Interactions' (No). A red box highlights the 'General Properties' section in the Help Panel.

46

The *Help panel* displays a short description of the property that you select in the properties panel. Extended Help can be found in the Help Topics selection from the Help main menu.

SAS Enterprise Miner: Interface Tour

The screenshot shows the SAS Enterprise Miner interface with the 'Diagram Workspace' highlighted by a red box. The workspace contains the same process flow as the previous screenshot: PVA97NK → Fill Replacement → Data Partition → Deploy → Regression. The Project Explorer on the left shows nodes like 'ORGANICS' and 'PVA97NK'. The Help Panel is also visible at the bottom.

47

In the *diagram workspace*, process flow diagrams are built, edited, and run. The workspace is where you graphically sequence the tools that you use to analyze your data and generate reports.

SAS Enterprise Miner: Interface Tour

Diagram Predictive Analysis opened
sasdemo@SASBAP as sasdemo Connected to SASApp - Logical Workspace Server (sasapp.demo.sas.com)

48

A process flow contains several nodes. *Nodes* are SAS Enterprise Miner tools that are connected by arrows to show the direction of information flow in an analysis.

SAS Enterprise Miner: Interface Tour

Diagram Predictive Analysis opened
sasdemo@SASBAP as sasdemo Connected to SASApp - Logical Workspace Server (sasapp.demo.sas.com)

49

The SAS Enterprise Miner tools that are available to your analysis are contained in the *tools palette*. The tools palette is arranged according to a process for data mining, SEMMA.

SEMMA is an acronym for the following words:

Sample - You sample the data by creating one or more data tables. The samples should be large enough to contain the significant information, but small enough to process.

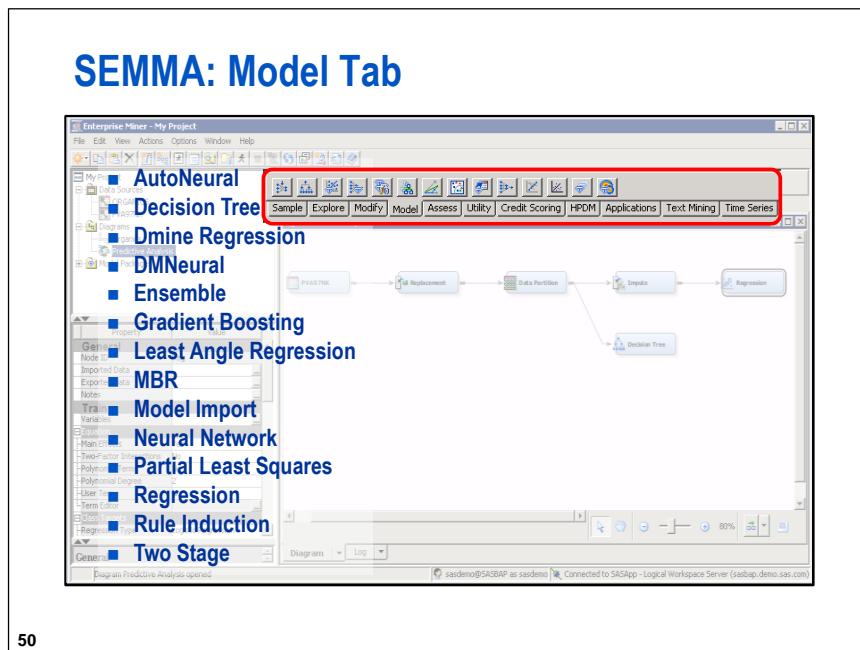
Explore - You explore the data by searching for anticipated relationships, unanticipated trends, and anomalies in order to gain understanding and ideas.

Modify - You modify the data by creating, selecting, and transforming the variables to focus the model selection process.

Model - You model the data by using the analytical tools to search for a combination of the data that reliably predicts a desired outcome.

Assess - You assess competing predictive models. (You build charts to evaluate the usefulness and reliability of the findings from the data mining process.)

Additional tools are available in the Utility group. There are also specialized group tools, namely, HPDM (High-Performance Data Mining), Applications, and Time Series. With additional licensing, Credit Scoring and Text Mining groups are also available. All tool groups are discussed on the next several pages.



The Model tab is a good starting location for new SAS Enterprise Miner users. The *Regression* tool enables you to fit both linear and logistic regression models to your data. You can use continuous, ordinal, and binary target variables. You can use both continuous and discrete variables as inputs. The tool supports the stepwise, forward, and backward selection methods. The interface enables you to create higher-order modeling terms such as polynomial terms and interactions.

Beyond SEMMA: Utility Tab

The screenshot shows the SAS Enterprise Miner interface with the title "Enterprise Miner - My Project". The toolbar at the top has several icons, and the "Utility" tab is highlighted with a red box. To the right of the toolbar, a list of utility tools is provided:

- Control Point
- End Groups
- Ext Demo
- Metadata
- Open Source Integration
- Register Model
- Reporter
- SAS Code
- Save Data
- Score Code Export
- Start Groups

The main workspace shows a process flow diagram with nodes like "PVA97NK", "File Replacement", "Deploy", and "Regression". On the left, there is a "Property" panel for the "PVA97NK" node.

51

The *Open Source Integration* tool enables you to write code in the R language inside SAS Enterprise Miner. The tool makes SAS Enterprise Miner data and metadata available to your R code and returns R results to SAS Enterprise Miner. In addition to training and scoring supervised and unsupervised R models, the Open Source Integration node enables data transformation and data exploration.

The *SAS Code* tool enables you to incorporate new or existing SAS code into process flow diagrams. The ability to write SAS code enables you to include additional SAS procedures into your data mining analysis. You can also use a SAS DATA step to create customized scoring code, to conditionally process data, and to concatenate or merge existing data sets. The tool provides a macro facility to dynamically reference data sets that are used for training, validation, testing, or scoring variables, such as input, target, and predict variables. After you run the SAS Code tool, the results and the data sets can then be exported for use by subsequent tools in the diagram.

Open Source Integration Node

- The Open Source Integration node enables the writing and use of R code in Enterprise Miner.
- It transfers data and results automatically between Enterprise Miner and R.
- All R packages must be installed in R before using the Open Source Integration node.



52

Open Source Integration Node Requirements

The Open Source Integration node requires you to

- enable R language statements in the SASV9.cfg file



- match the appropriate versions of SAS, R, and the PMML R package.

SAS EM Version	R Version	PMML Version
13.1	2.13.0 – 3.0.2	pmml_1.4.1
13.2	2.15.3 – 3.0.3	pmml_1.4.1
14.1	3.0.1 – 3.1.2	pmml_1.4.2

53

The Open Source Integration node is verified to work with 64-bit R. (32-bit R is not recommended.)

The SYSCC=10 error indicates that the appropriate versions of SAS, R, and PMML are not being used. The user should uninstall the PMML package (or R, or both) and download the appropriate version.

Output Mode: PMML

Predictive Modeling Markup Language (PMML) is an open standard that enables certain R models to be translated into SAS DATA step code.

Here are the currently supported R models:

- linear models (lm)
- multinomial log-linear models (multinom)
- generalized linear models (glm)
- decision trees (rpart)
- neural networks (nnet)
- K-means clustering (kmeans)

Train	
Variables	...
Code Editor	...
Language	R
Training Mode	Supervised
Output Mode	PMML

54

The Output mode specifies different ways that the output from the R code is available. Options are PMML, Merge, or None. Output mode **None** is used primarily to debug the R code and ensure that it is working properly. The log provides more detail about errors when output mode **None** is specified.

Output Mode: Merge

Merging the output mode enables integration with the thousands of R packages that are not supported in the PMML output mode.

- Variables created in R are merged with the SAS Enterprise Miner data source by the user.
- SAS DATA step code is not created.

Train	
Variables	...
Code Editor	...
Language	R
Training Mode	Supervised
Output Mode	Merge

55

The Merge mode is commonly used when applying the **predict()** function to the R model object. The **predict()** function returns results and merges the results to the workflow data set.

R Script: Variable Handles

Enterprise Miner variable handles are used to efficiently create an R script. Here are the variable handles:

- &EMR_MODEL
 - refers to the R model object.
- &EMR_NUM_TARGET and &EMR_CLASS_TARGET
 - refer to the response variable.
- &EMR_NUM_INPUT and &EMR_CLASS_INPUT
 - refer to the input variables.
- &EMR_IMPORT_DATA
 - refers to the workflow data set.

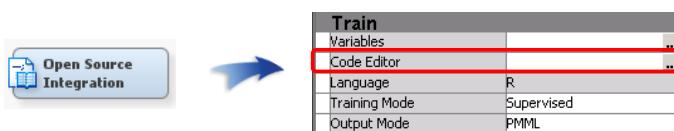
56

The words NUM and CLASS in a variable handle refer to numeric or categorical variables. A single INPUT variable handle refers to the entire set of numeric or categorical variables to be used as inputs.

The &EMR_MODEL variable handle is used to translate the R object into SAS DATA step code using PMML.

Example R Script

Select the Code Editor ellipsis to create the R script.



- Without Variable Handles:

```
&EMR_MODEL <- lm(rY ~ X1 + X2 + X3 +
C1 + C2 + C3, data =&EMR_IMPORT_DATA)
```

- With Variable Handles:

```
&EMR_MODEL <- lm(&EMR_NUM_TARGET ~
&EMR_NUM_INPUT + &EMR_CLASS_INPUT,
data=&EMR_IMPORT_DATA)
```

57

The &EMR_MODEL and &EMR_IMPORT_DATA variable handles must be used.

8.03 Poll

Variable handles must be used in SAS Enterprise Miner.

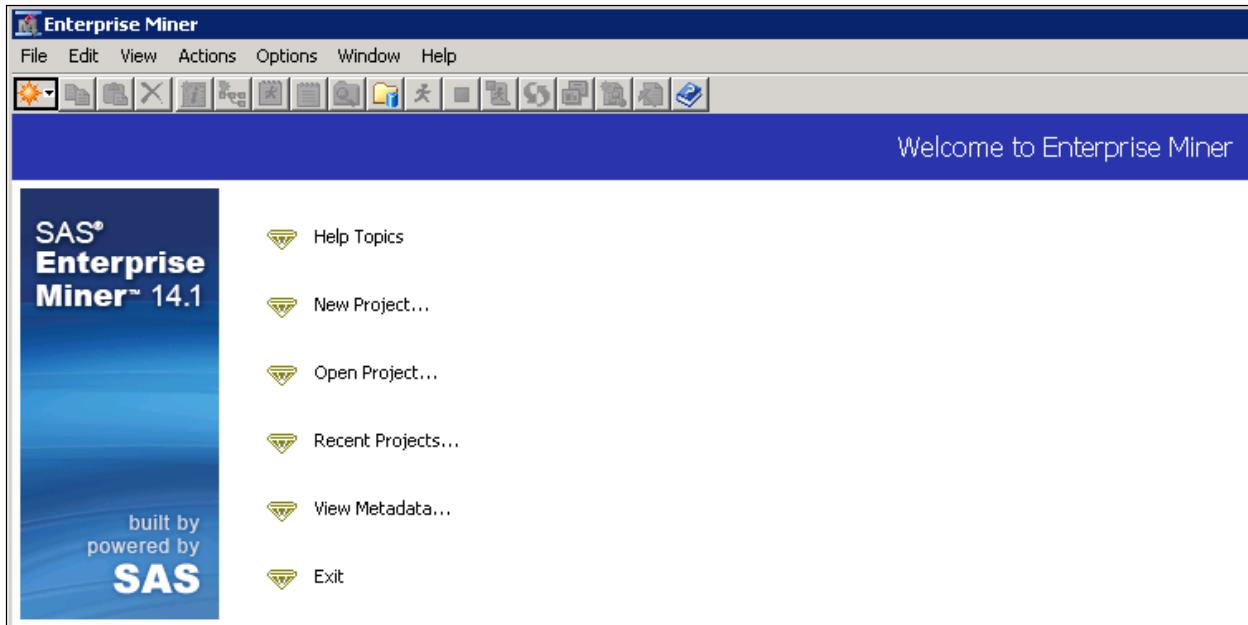
- True
- False



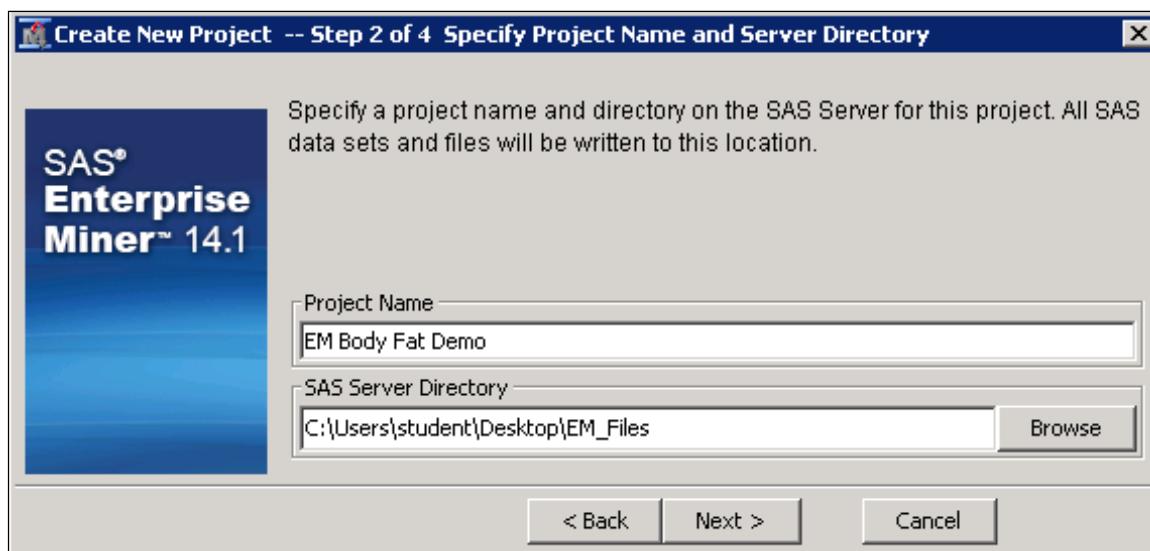
Getting Started with Enterprise Miner and the Open Source Integration Node

Percentage of body fat, age, weight, height, and 10 body circumference measurements were recorded for 252 men by Dr. Roger W. Johnson of Calvin College in Minnesota. The data are in the **BodyFat** data set. The following variables are in the data set:

Case	Case Number
PctBodyFat1	Percent body fat
PctBodyFat2	Percent body fat using Siri's equation, $(495/\text{Density}) - 450$
Density	Body density
Age	Age in years
Weight	Weight in pounds
Height	Height in inches
Adioposity	A BMI measure
FatFreeWt	Body weight with no fat
Neck	Neck circumference (cm)
Chest	Chest circumference (cm)
Abdomen	Abdomen circumference (cm)
Hip	Hip circumference (cm)
Thigh	Thigh circumference (cm)
Knee	Knee circumference (cm)
Ankle	Ankle circumference (cm)
Biceps	Extended biceps circumference (cm)
Forearm	Forearm circumference (cm)
Wrist	Wrist circumference (cm)

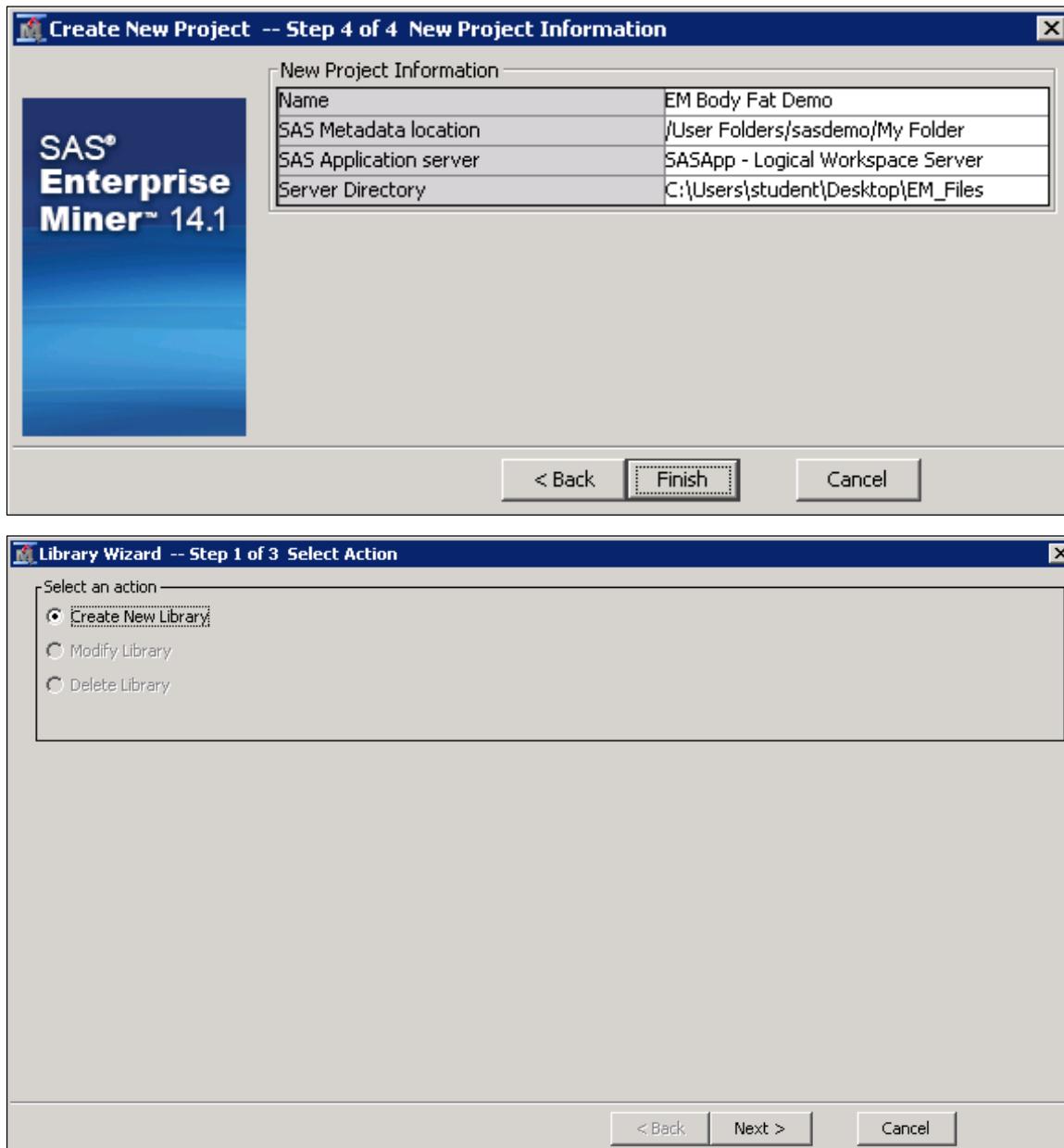


1. Open Enterprise Miner and click the **New Project** tab.
2. Click **Next** for Step 1.
3. At Step 2, enter the project name **EM Body Fat Demo** and let the SAS server directory be a user-created file on the desktop. Here it is called **EM_Files**. Click **Next**.



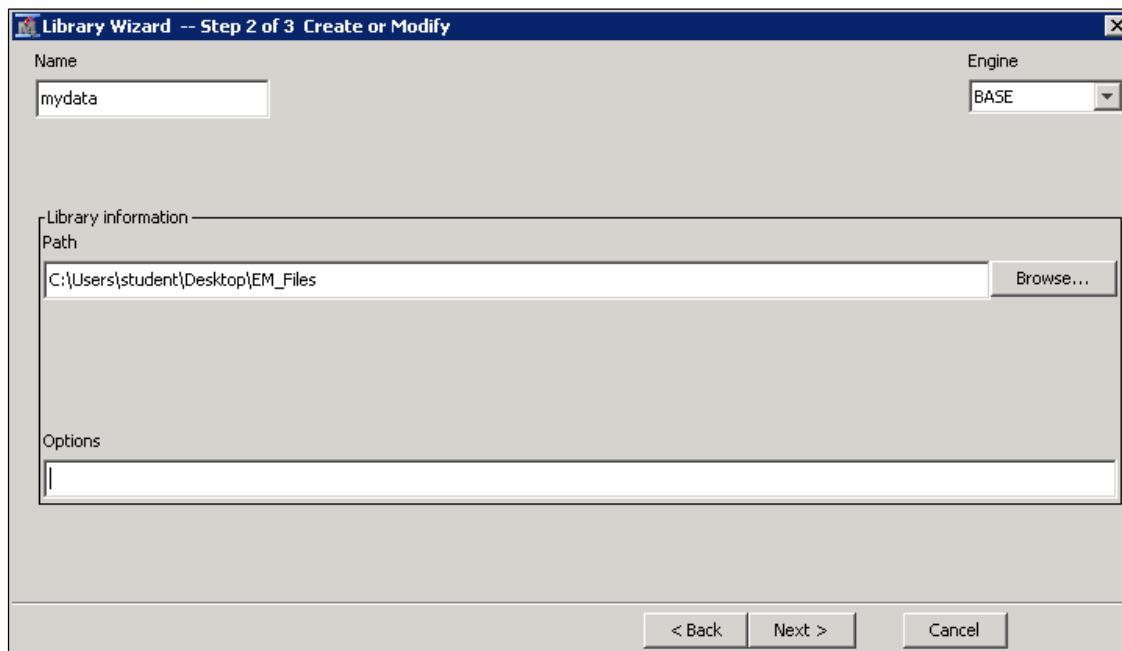
2. Click **Next** at Step 3.

3. Click **Finish** at Step 4 to create the new project.

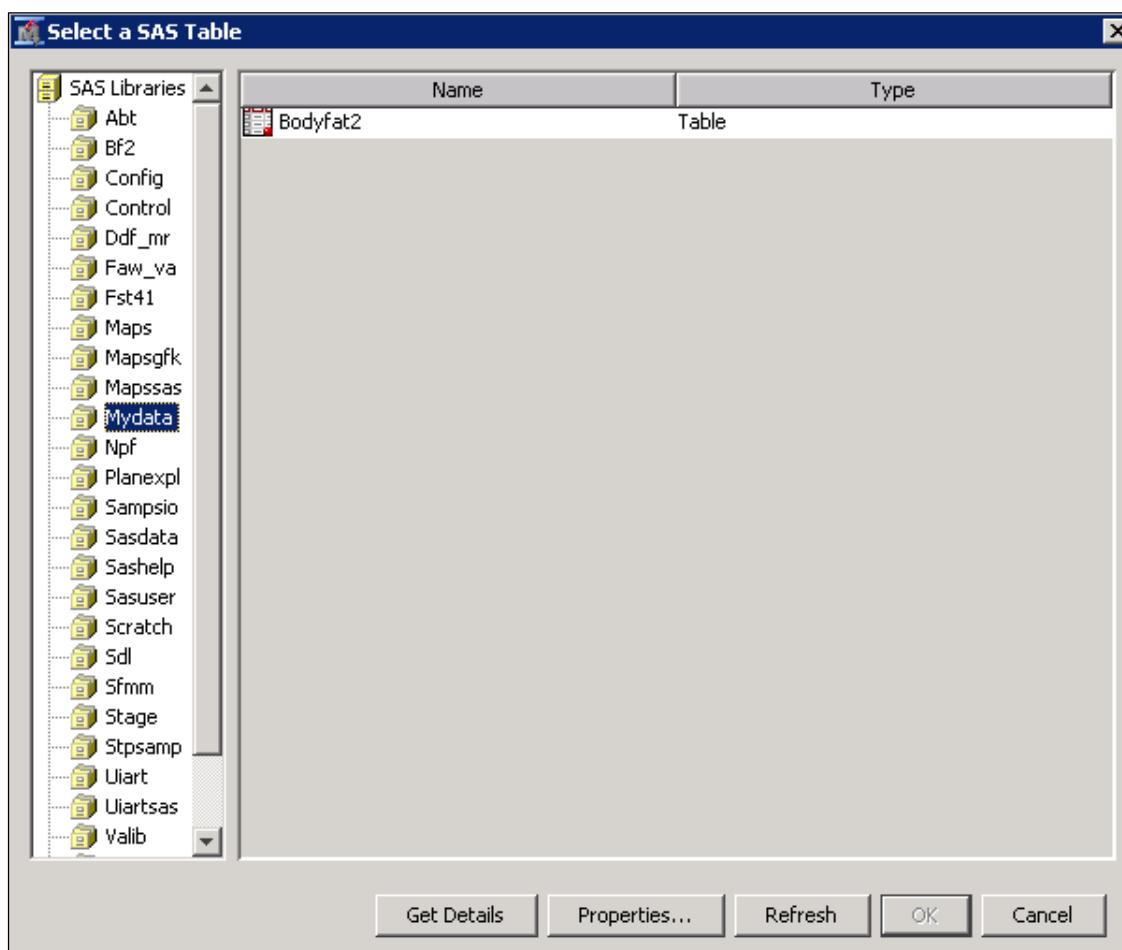


4. Select **File** \Rightarrow **New** \Rightarrow **Library** and click **Next**. Creating the library gives the user access to saved data sets.

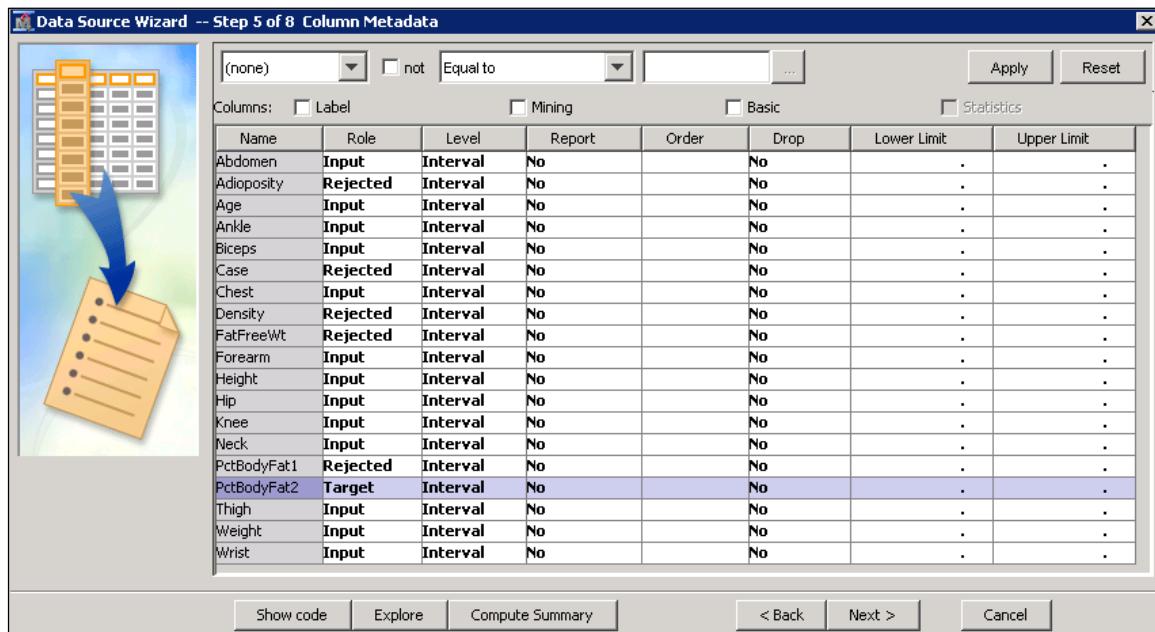
5. Name the new library **mydata** and set the path to the **Bodyfat2** data set.



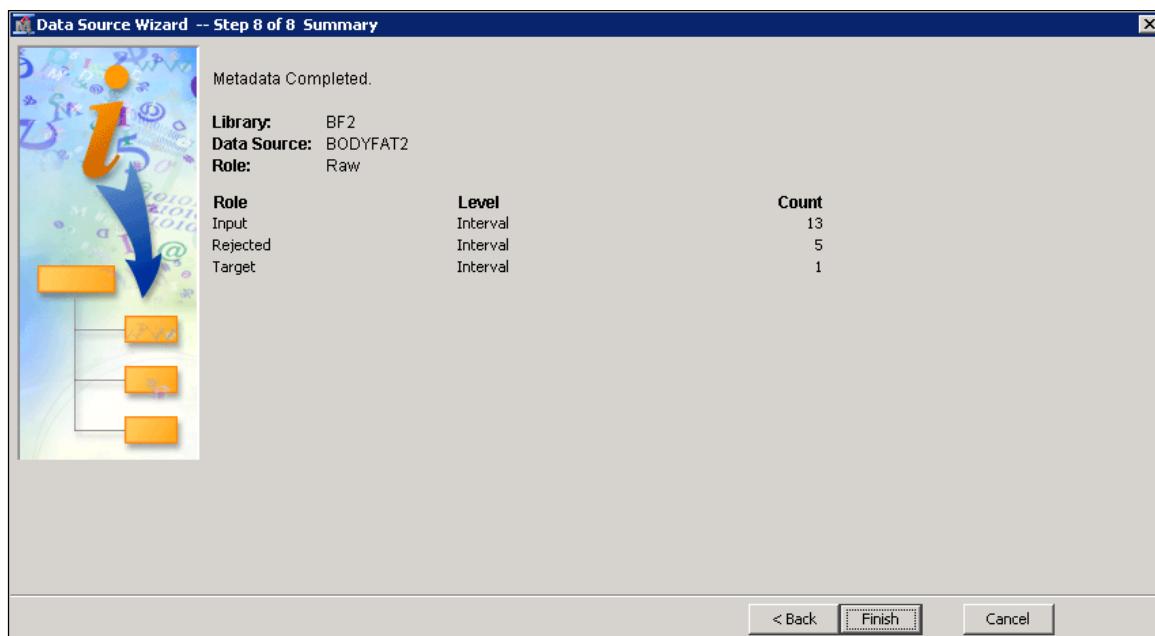
6. Click **Next** and then click **Finish** on Step 3.

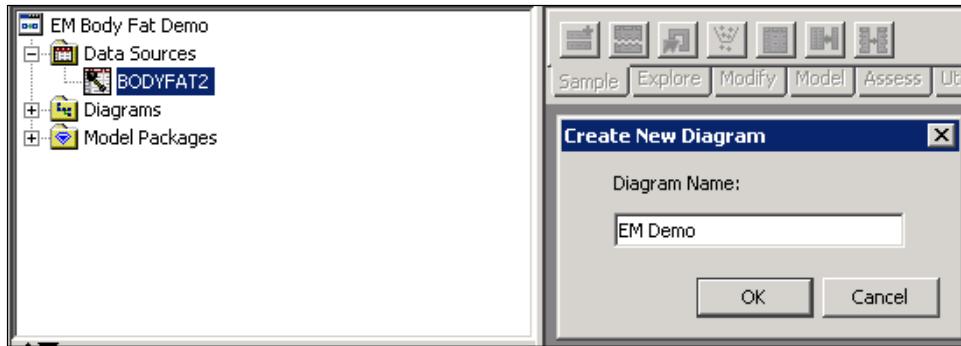


7. Select **File** \Rightarrow **New** \Rightarrow **Data Source** and then click **Next** at Step 1.
8. At Step 2, select **Browse** and select the **Bodyfat2** data set from the **mydata** library that was created.
9. Click **OK** and then click **Next** for Steps 3 and 4.
10. At Step 5, change the role of **Adioposity**, **Case**, **Density**, **FatFreeWt**, and **PctBodyFat1** to **Rejected**. This excludes the variables from subsequent analyses. Change the role of **PctBodyFat2** to **Target**. This keeps the **PctBodyFat2** variable fixed as the response variable for subsequent analyses. All other variables are considered inputs, or predictor variables. The level is not changed because all variables are continuous (or interval). Click **Next**.

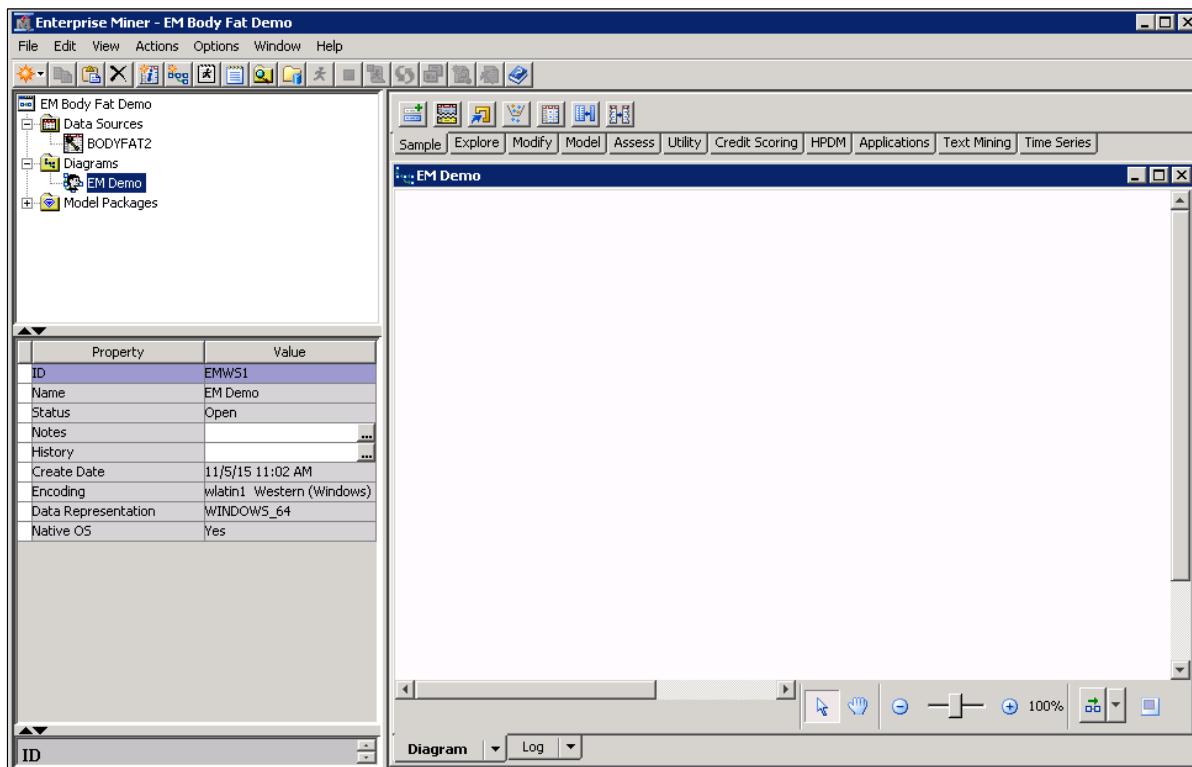


11. Click **Next** through to Step 8 and then click **Finish**. There should be 13 input variables, five rejected variables, and only one target variable.





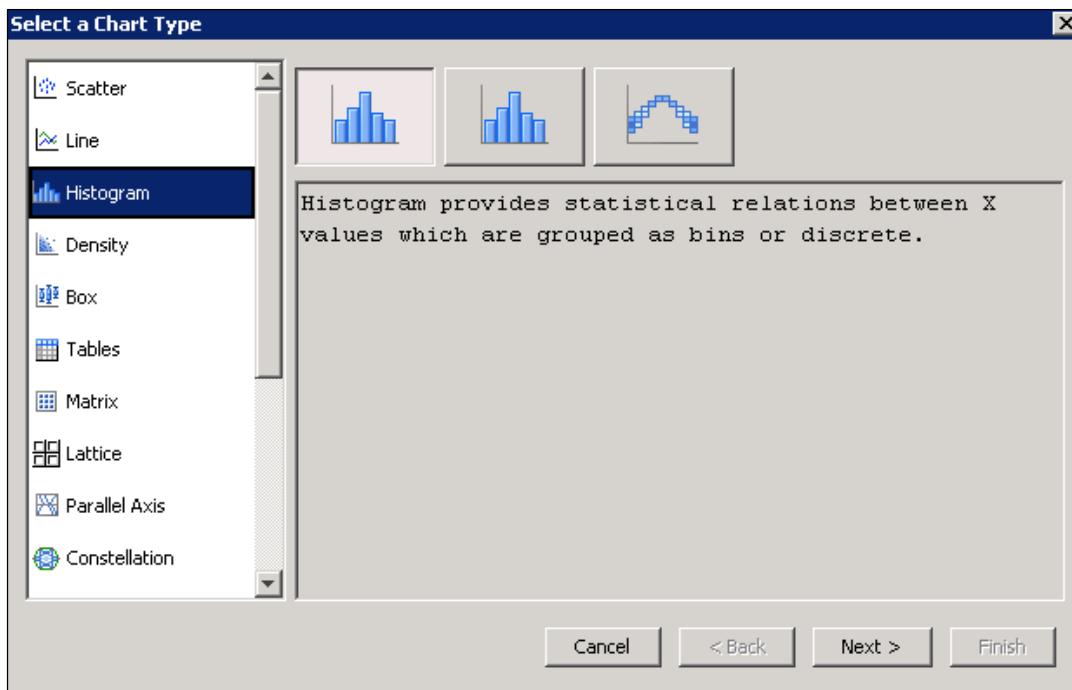
12. Notice that the **BODYFAT2** data set is now available in the projects panel.
13. Select **File** \Rightarrow **New** \Rightarrow **Diagram** and enter **EM Demo** as the diagram name. Click **OK**.



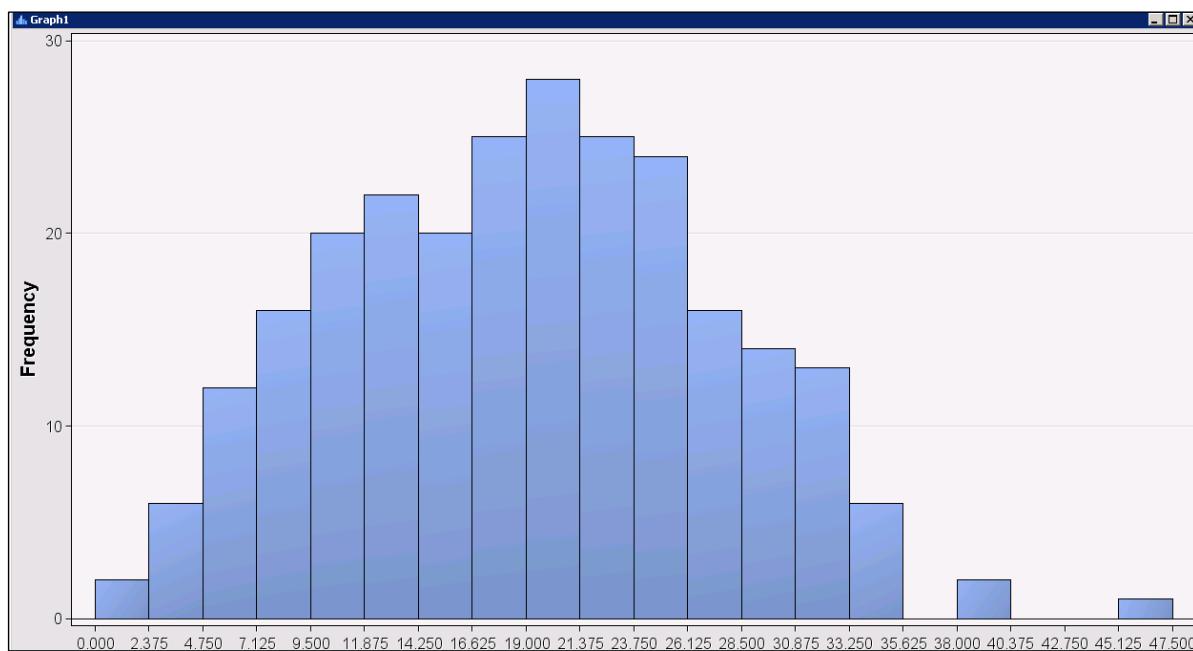
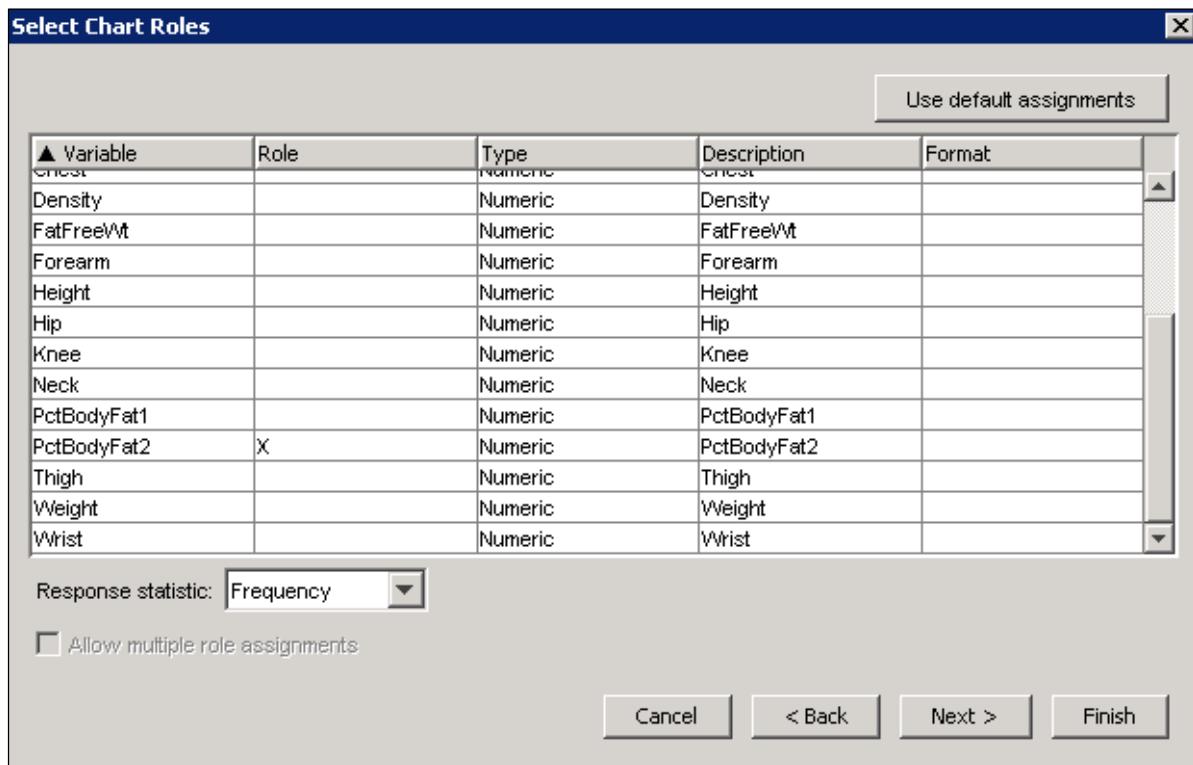
14. The workspace is now ready for the user to create a process flow. Before you create the process flow, explore the data set variables. Right-click the data set in the projects panel and select **Explore**.

Obs #	Variable ...	Label	Type	Percent ...	Minimum	Maximum	Mean
1	Abdomen		VAR	0	69.4	148.1	92.55595
2	Adioposity		VAR	0	18.1	48.9	25.4369
3	Age		VAR	0	22	81	44.88492
4	Ankle		VAR	0	19.1	33.9	23.10238
5	Biceps		VAR	0	24.8	45	32.27341
6	Case		VAR	0	1	252	126.5
7	Chest		VAR	0	79.3	136.2	100.8242
8	Density		VAR	0	0.995	1.1089	1.055574
9	FatFreeWt		VAR	0	105.9	240.5	143.7139
10	Forearm		VAR	0	21	34.9	28.66389
11	Height		VAR	0	64	77.75	70.30754
12	Hip		VAR	0	85	147.7	99.90476
13	Knee		VAR	0	33	49.1	38.59048
14	Neck		VAR	0	31.1	51.2	37.99206
15	PctBodyFat1		VAR	0	0	45.1	18.93849
16	PctBodyFat2		VAR	0	0	47.5	19.15079
17	Thigh		VAR	0	47.2	87.3	59.40595
18	Weight		VAR	0	118.5	363.15	178.9244
19	Wrist		VAR	0	15.8	21.4	18.22976

15. Notice that the sample statistics for each variable are in the upper right portion of the window and the data set itself is printed in the bottom panel.
16. To create a histogram of the target variable, select **Actions** \Rightarrow **Plot** and then select **Histogram**. Click **Next**.

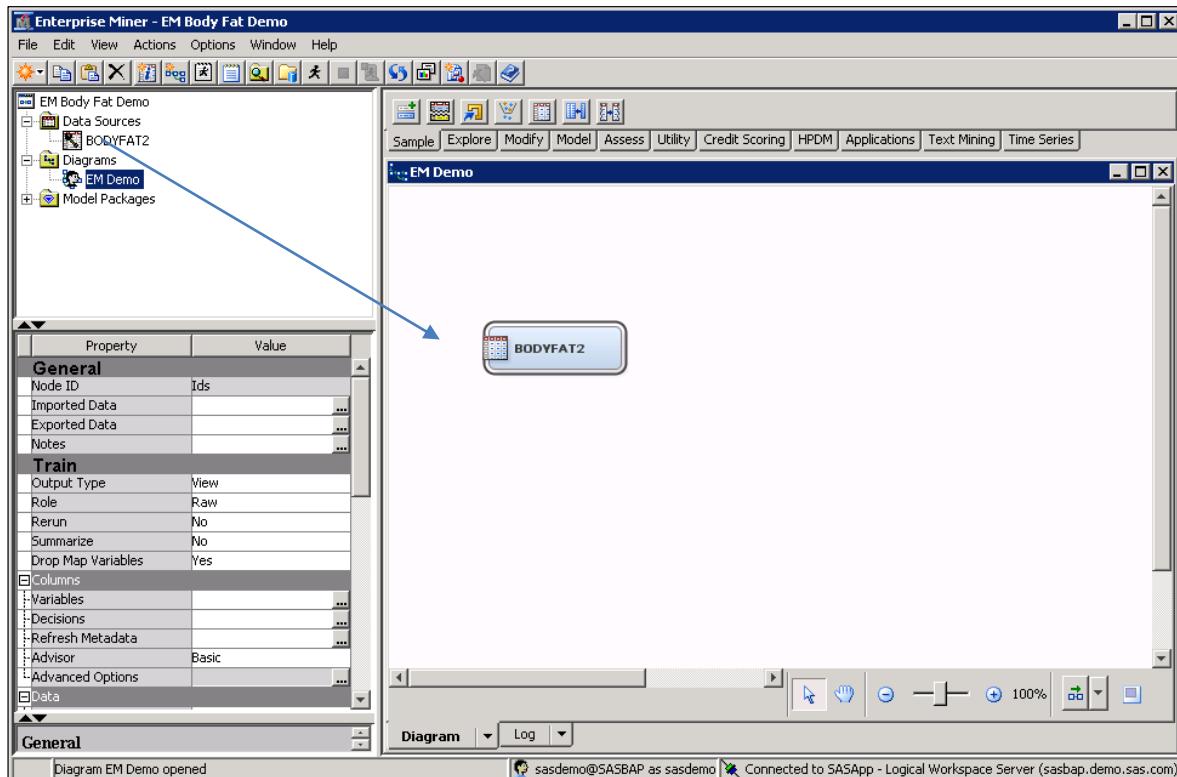


17. Change the role of the **PctBodyFat2** variable to **X** and click **Finish**.



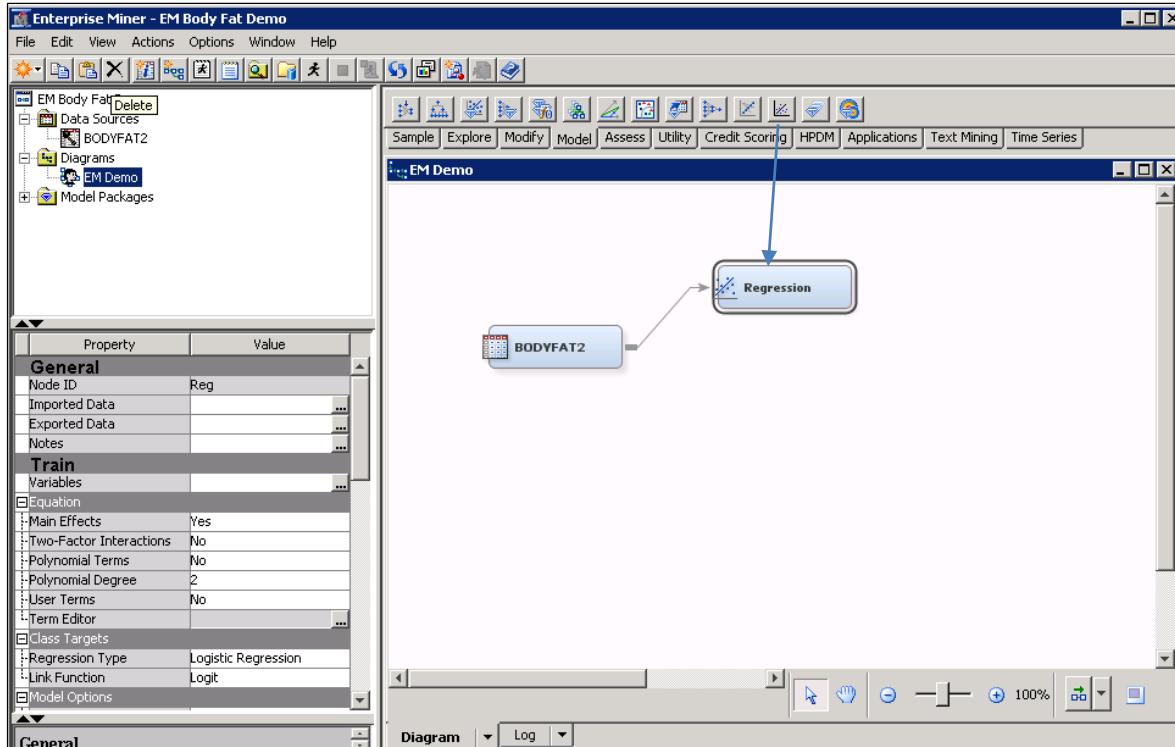
18. To change the number of bins, select **View** \Rightarrow **Graph Properties** and change the number of bins to **20**. Close the Explore window when you are ready.

19. To begin the process flow, simply drag and drop the **BODYFAT2** data set into the diagram from the project panel. Notice that the properties panel changes when the user selects a node.

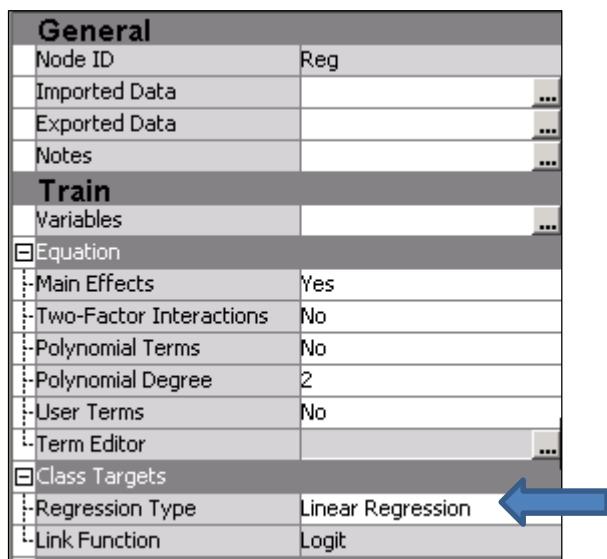


20.

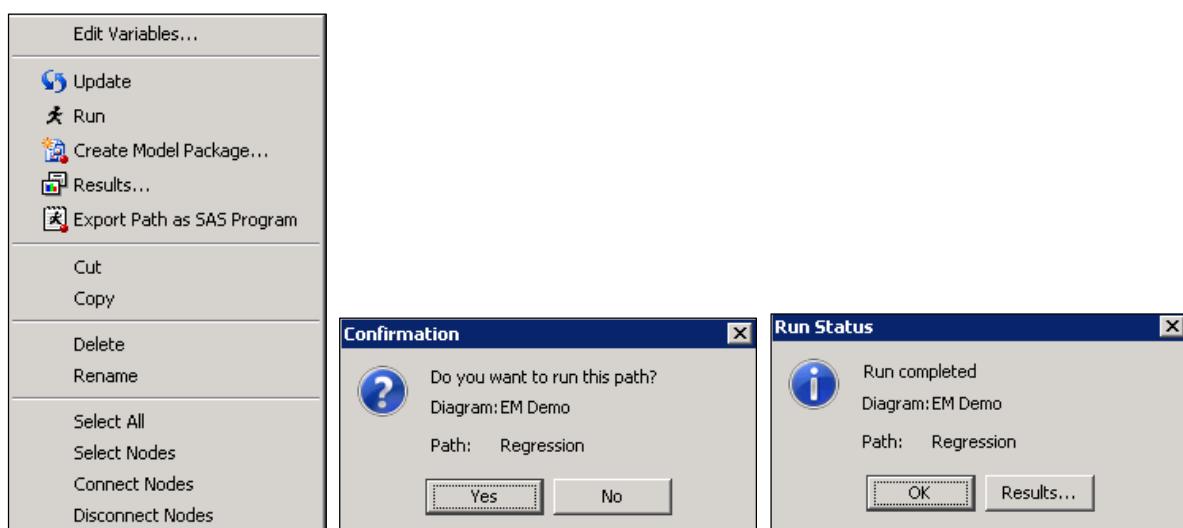
21. Click the **Model** tab and drag the **Regression** node into the diagram. Simply connect the two nodes to create a process flow.

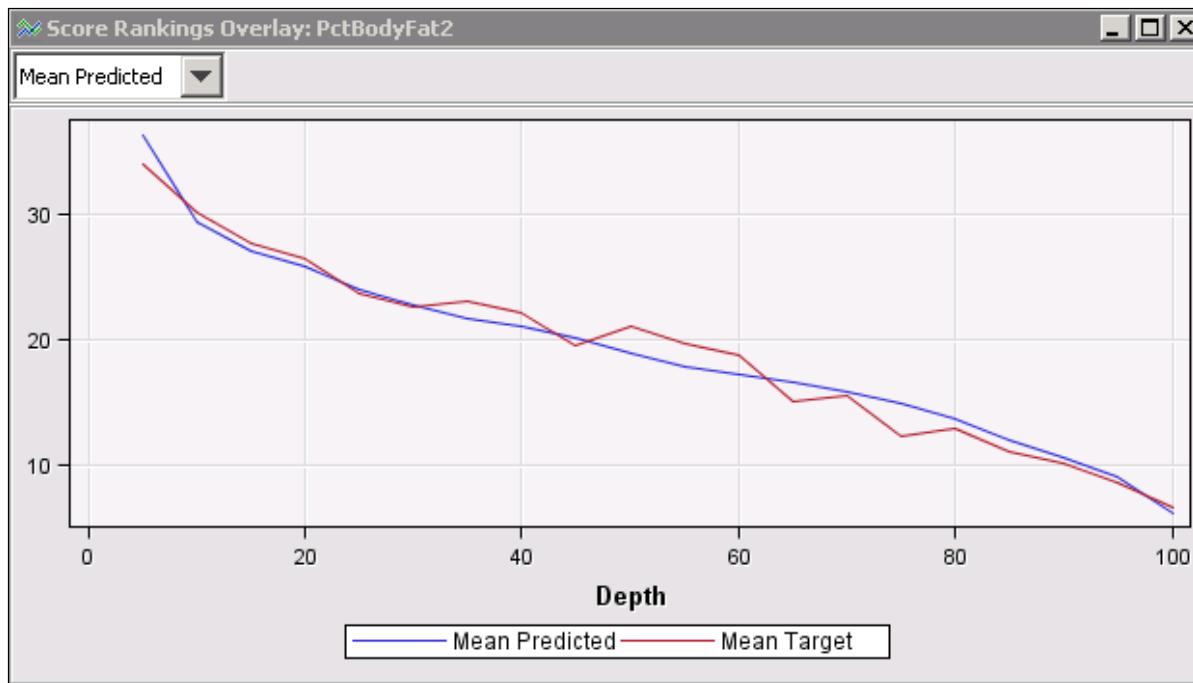


22. Change the Regression Type value under Class Targets from **Logistic Regression** to **Linear Regression** because the target variable is continuous.



23. To run the regression analysis, right-click the **Regression** node and select **Run**. Click **Yes** in the confirmation window. Click **Results** to view the output. Notice that each node becomes green when it runs successfully.

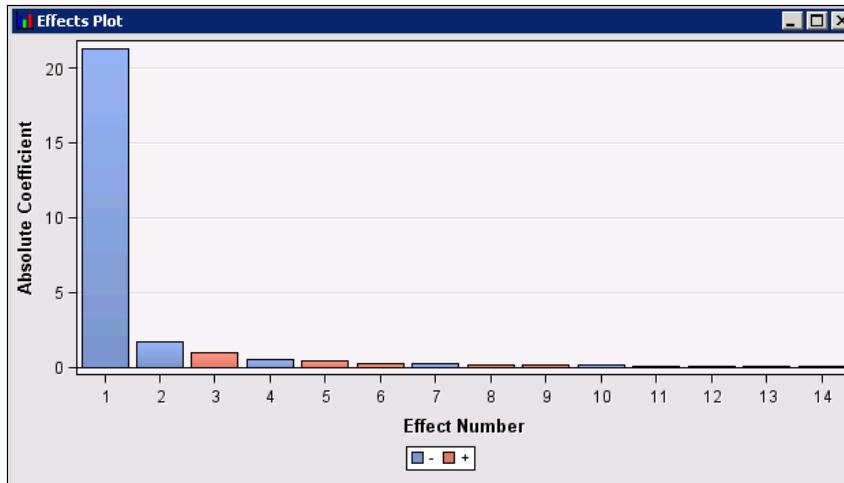




The Score Rankings Overlay plot charts the true target value and the predicted target value from the model. The depth on the X axis ranges from 0 to 100 and refers to the percentage of values. For example, the depth from 0 to 20 refers to the highest 20% of predicted values. Thus, the depth sorts the target from largest to smallest.

Target	Target Label	Fit Statistics	Statistics Label	Train	Validation	Test
PctBodyFat2		_AIC_	Akaike's Information Criterion	749.8491	.	.
PctBodyFat2		_ASE_	Average Squared Error	17.53994	.	.
PctBodyFat2		_AVERR_	Average Error Function	17.53994	.	.
PctBodyFat2		_DFE_	Degrees of Freedom for Error	238	.	.
PctBodyFat2		_DFM_	Model Degrees of Freedom	14	.	.
PctBodyFat2		_DFT_	Total Degrees of Freedom	252	.	.
PctBodyFat2		_DIV_	Divisor for ASE	252	.	.
PctBodyFat2		_ERR_	Error Function	4420.064	.	.
PctBodyFat2		_FPE_	Final Prediction Error	19.60346	.	.
PctBodyFat2		_MAX_	Maximum Absolute Error	11.19662	.	.
PctBodyFat2		_MSE_	Mean Square Error	18.5717	.	.
PctBodyFat2		_NOBS_	Sum of Frequencies	252	.	.
PctBodyFat2		_NW_	Number of Estimate Weights	14	.	.
PctBodyFat2		_RASE_	Root Average Sum of Squares	4.188071	.	.
PctBodyFat2		_RFPE_	Root Final Prediction Error	4.427579	.	.
PctBodyFat2		_RMSE_	Root Mean Squared Error	4.309489	.	.
PctBodyFat2		_SBC_	Schwarz's Bayesian Criterion	799.2611	.	.
PctBodyFat2		_SSE_	Sum of Squared Errors	4420.064	.	.
PctBodyFat2		_SUMW_	Sum of Case Weights Times Freq	252	.	.

The Fit Statistics panel provides a range of statistics associated with the model fit.



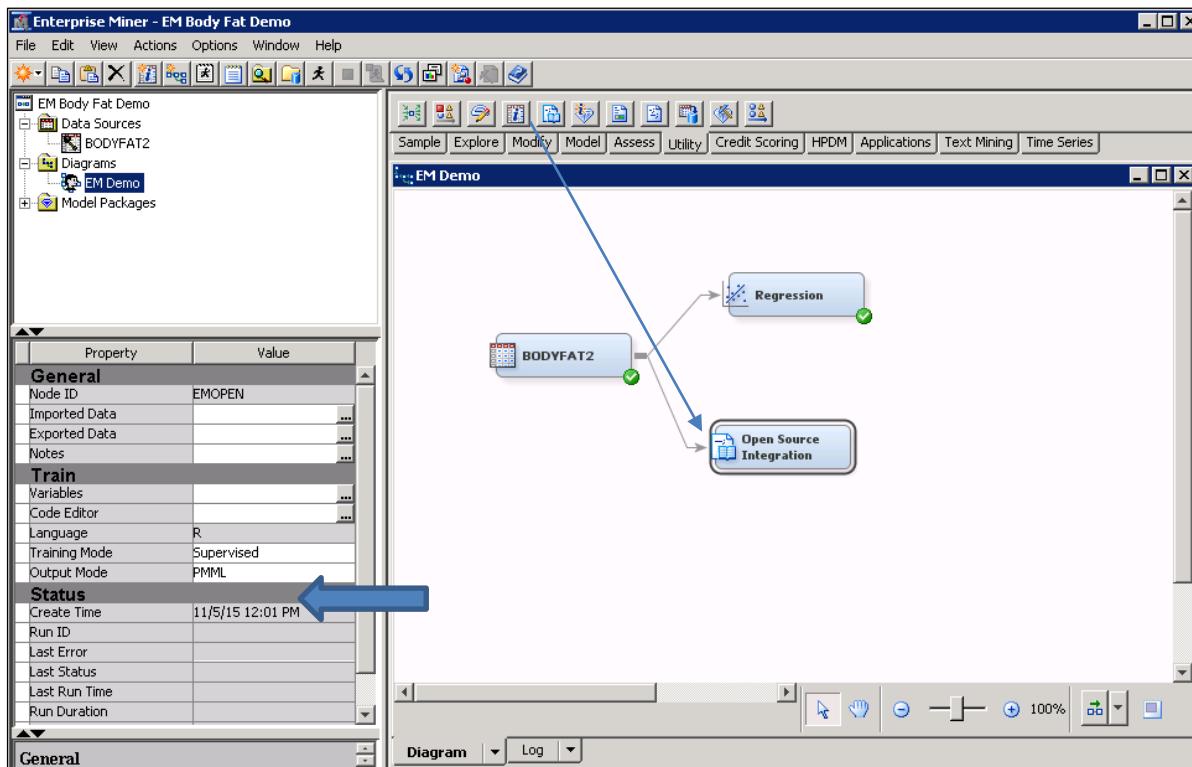
The Effects plot provides a bar chart for each model coefficient. The absolute value is plotted, and the sign of the estimate is color coded.

Partial Output

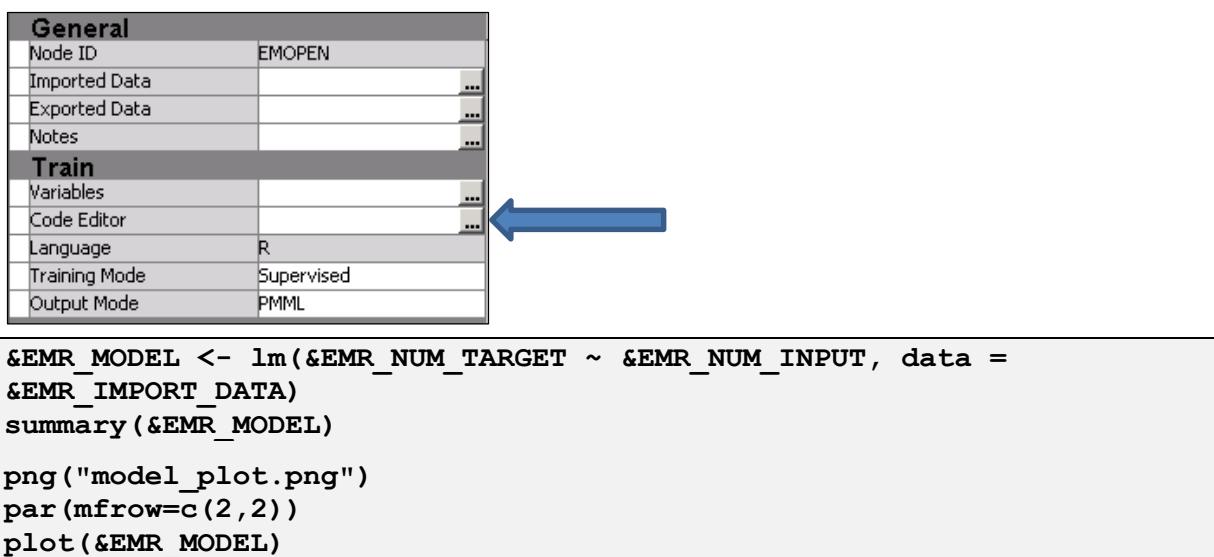
Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	13	13159	1012.225064	54.50	<.0001
Error	238	4420.064012	18.571698		
Corrected Total	251	17579			
Model Fit Statistics					
R-Square	0.7486	Adj R-Sq	0.7348		
AIC	749.8491	BIC	753.4892		
SBC	799.2611	C(p)	14.0000		
Analysis of Maximum Likelihood Estimates					
Parameter	DF	Estimate	Standard Error	t Value	Pr > t
Intercept	1	-21.3532	22.1862	-0.96	0.3368
Abdomen	1	0.9550	0.0902	10.59	<.0001
Age	1	0.0646	0.0322	2.01	0.0460
Ankle	1	0.1779	0.2226	0.80	0.4251
Biceps	1	0.1823	0.1725	1.06	0.2917
Chest	1	-0.0172	0.1032	-0.17	0.8679
Forearm	1	0.4557	0.1993	2.29	0.0231
Height	1	-0.0439	0.1787	-0.25	0.8060
Hip	1	-0.1886	0.1448	-1.30	0.1940
Knee	1	0.0139	0.2477	0.06	0.9552
Neck	1	-0.4755	0.2356	-2.02	0.0447
Thigh	1	0.2483	0.1462	1.70	0.0906
Weight	1	-0.0964	0.0618	-1.56	0.1205
Wrist	1	-1.6545	0.5332	-3.10	0.0021

The Output panel provides output associated with the model. For the Regression node, it provides the ANOVA table, model fit statistics, and parameter estimates. It also provides the same summary as the Fit Statistics panel and gives the predicted value for every five percentage points of depth.

24. Close the Results node when you are ready.
25. To supplement Enterprise Miner tools with the open source software R, drag and drop the **Open Source Integration** node into the diagram from the Utility tab. Connect the node to the **BODYFAT2** data source.

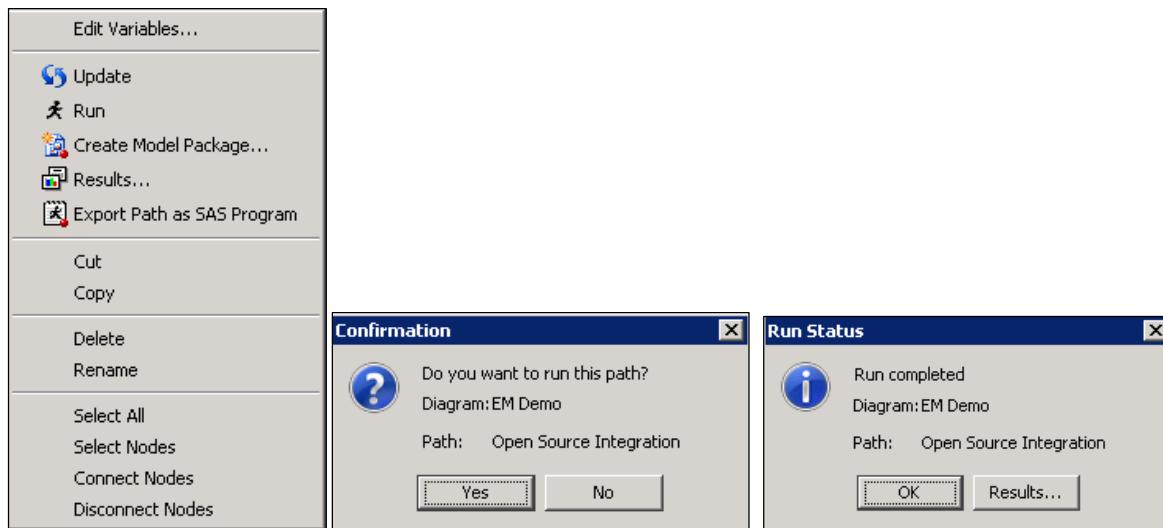


26. Notice that the Output Mode value is set to **PMML** by default.
27. Select the ellipsis next to the Code Editor and add the R script shown below. Click **OK** after you create the script.



28. Notice that the script uses SAS variable handles. The **&EMR_NUM_INPUT** variable handles the user from entering the 13 input variables.

29. To run the R script, right-click the **Regression** node and select **Run**. Click **Yes** in the confirmation window. Click **Results** to view the output. Notice that the Open Source Integration node changes to green after it runs successfully.



The results provide the same Score Rankings Overlay plot but no Effects plot.



Notice that the Fit Statistics panel provides fewer statistics than the results from the Regression node.

Target	Target Label	Fit Statistics	Statistics Label	Train
PctBodyFat2		_ASE_	Average Squared Error	17.53994
PctBodyFat2		_DIV_	Divisor for ASE	252
PctBodyFat2		_MAX_	Maximum Absolute Error	11.19662
PctBodyFat2		_NOBS_	Sum of Frequencies	252
PctBodyFat2		_RASE_	Root Average Squared Error	4.188071
PctBodyFat2		_SSE_	Sum of Squared Errors	4420.064

R Handle Contents		
Handle	Available	Contents
EMR_NUM_TARGET	Y	rPctBodyFat2
EMR_CLASS_TARGET	Y	
EMR_NUM_INPUT	Y	Abdomen + Age + Ankle + Biceps + Chest + Forearm + Height + Hip + Knee + Neck + Thigh + Weight + Wrist
EMR_CLASS_INPUT	Y	
EMR_CHAR_INPUT	Y	

```
Call:
lm(formula = rPctBodyFat2 ~ Abdomen + Age + Ankle + Biceps +
   Chest + Forearm + Height + Hip + Knee + Neck + Thigh + Weight +
   Wrist, data = EMR_IMPORT_DATA)
```

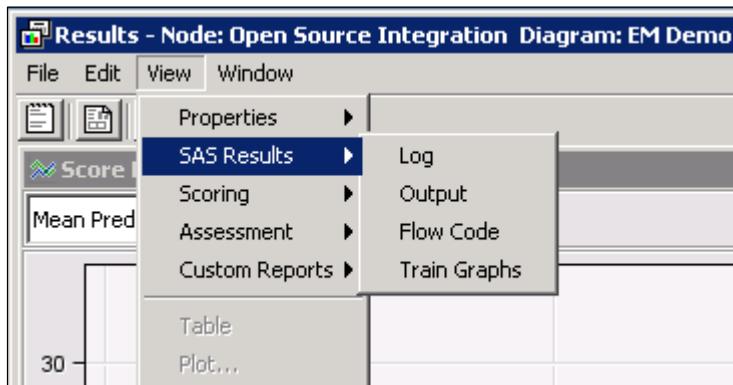
The Output panel translates the SAS variable handles into the appropriate R code and prints it to the Output log. For example, &EMR_NUM_INPUT was changed to Abdomen + Age + Ankle + Biceps + Chest + Forearm + Height + Hip + Knee + Neck + Thigh + Weight + Wrist. The target and data source were also altered to run in R.

Coefficients:					
	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-21.35323	22.18616	-0.962	0.33680	
Abdomen	0.95500	0.09016	10.592	< 2e-16 ***	
Age	0.06457	0.03219	2.006	0.04601 *	
Ankle	0.17788	0.22262	0.799	0.42505	
Biceps	0.18230	0.17250	1.057	0.29166	
Chest	-0.01718	0.10322	-0.166	0.86792	
Forearm	0.45574	0.19930	2.287	0.02309 *	
Height	-0.04394	0.17870	-0.246	0.80599	
Hip	-0.18859	0.14479	-1.302	0.19401	
Knee	0.01395	0.24775	0.056	0.95516	
Neck	-0.47547	0.23557	-2.018	0.04467 *	
Thigh	0.24835	0.14617	1.699	0.09061 .	
Weight	-0.09638	0.06185	-1.558	0.12047	
Wrist	-1.65450	0.53316	-3.103	0.00215 **	

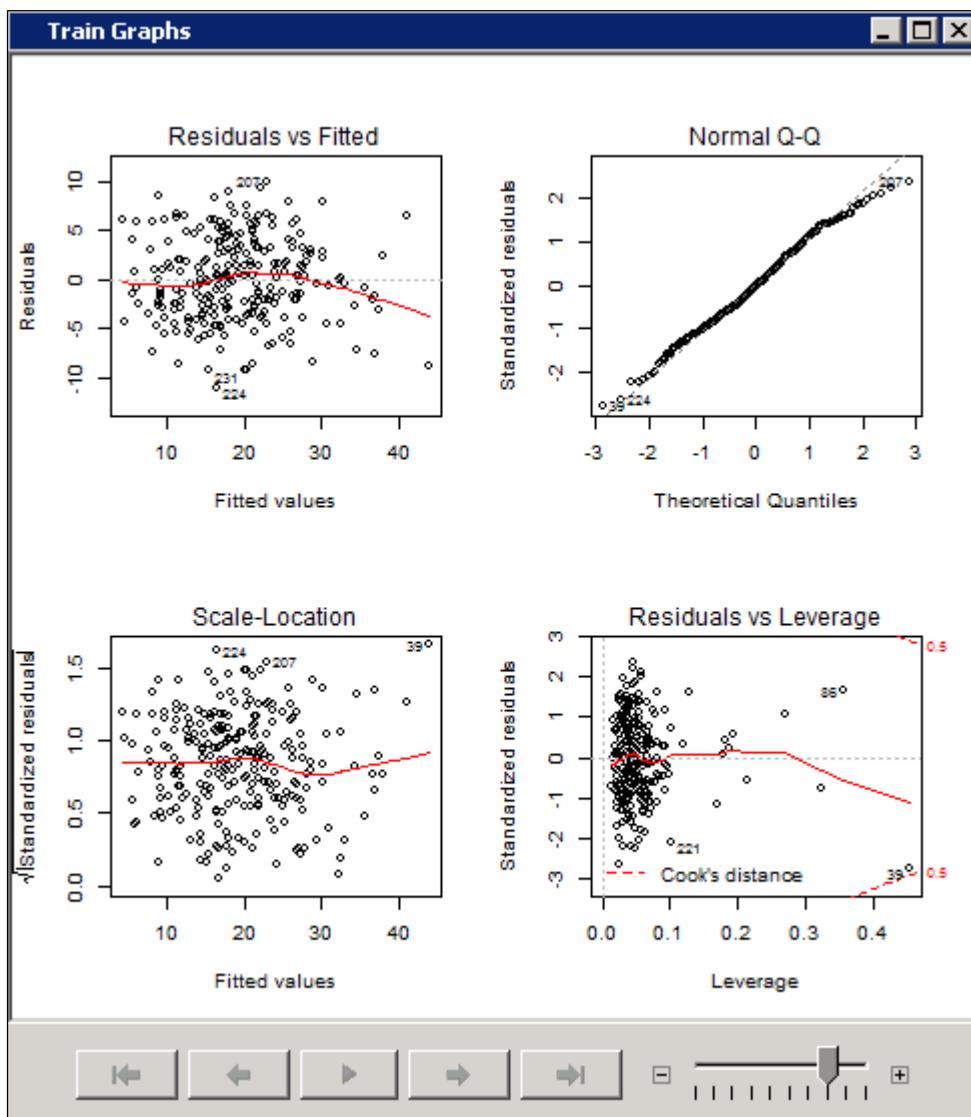
Signif. codes: 0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1					
Residual standard error: 4.309 on 238 degrees of freedom					
Multiple R-squared: 0.7486, Adjusted R-squared: 0.7348					
F-statistic: 54.5 on 13 and 238 DF, p-value: < 2.2e-16					

The second part of the output is the output from the R command line. This was generated from the **summary()** function in R. Just like in IML, all R command line output is returned and printed in the Enterprise Miner output log.

30. To view the **plot()** function R output, select **View** \Rightarrow **SAS Results** \Rightarrow **Train Graphs**.



The Train Graphs window saves the R plots. However, you must use the **png()** function to do so.



End of Demonstration

Applied Analytics Using SAS Enterprise Miner Course

Enterprise Miner is an incredibly powerful predictive modeling tool. The Applied Analytics Using SAS® Enterprise Miner™ course teaches you Enterprise Miner capabilities. You learn about the following features:

- how to train, validate, and test models
- decision trees
- the nuances of regression in predictive modeling
- neural networks
- pattern discovery
- model assessment
- and more!

8.4 Solutions

Solutions to Exercises

1. Comparing Multiple Regression Estimates in SAS and R

- a. Begin by invoking PROC IML and exporting the **fish** data set to R as a data frame with the name **Fish**.

```
proc iml;
  call ExportDataSetToR("sp4r.fish","fish");
```

- b. Fit a linear model with **Weight** as the dependent variable and **Height** and **Width** as the independent variables using the **lm()** function. Store the object and use the **summary()** function to print model estimates.

```
submit / r;
  fit <- lm(Weight ~ Height + Width, data=fish)
  summary(fit)
endsubmit;
```

```
Call:
lm(formula = Weight ~ Height + Width, data = fish)

Residuals:
    Min      1Q  Median      3Q     Max 
-249.90 -98.50 -46.03  57.34 890.57 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -433.653    37.020 -11.714 <2e-16 ***
Height       5.507     5.086   1.083   0.281    
Width        177.444   12.884  13.773 <2e-16 ***  
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 166 on 155 degrees of freedom
(1 observation deleted due to missingness)
Multiple R-squared:  0.7891,    Adjusted R-squared:  0.7864 
F-statistic: 290 on 2 and 155 DF,  p-value: < 2.2e-16
```

2.

- a. Import the parameter estimates into an IML matrix. Recall that the parameter estimates are stored under the name **Coefficients** in the R object.

```
call ImportMatrixFromR(r_Coefficients,"fit$coefficients");
```

- b. Run the same analysis in SAS using PROC REG. Output the parameter estimates using the OUTEST= option in the PROC REG statement.

```
submit;
  ods select none;
  proc reg data=sp4r.fish outest=sp4r.betas;
    model weight = height width;
  run; quit;
  ods select default;
endsubmit;
```

- c. Import the SAS coefficients into IML using the USE, READ, and CLOSE statements.

```
use sp4r.betas;
read all var {intercept height width} into sas_Coefficients;
close sp4r.betas;
```

- d. Print the SAS coefficients and R coefficients side by side along with the difference between the estimates.

```
coefficients = sas_coefficients` || r_coefficients ||
  (sas_coefficients` - r_coefficients);

reset noname;
coefficientNames = {SAS_Coefficients R_Coefficients Difference};
print coefficients[colname=coefficientNames];
quit;
```

SAS_COEFFICIENTS	R_COEFFICIENTS	DIFFERENCE
-433.6525	-433.6525	-5.12E-13
5.5068475	5.5068475	-7.66E-13
177.44357	177.44357	1.648E-12

End of Solutions

Solutions to Student Activities (Polls/Quizzes)

8.01 Multiple Answer Poll – Correct Answers

Choose the correct statements.

- a. R can be called from Base SAS, SAS/IML, and SAS Enterprise Miner.
- b. -RLANG must be added to the SAS configuration file.
- c. PROC OPTIONS is used to test the SAS and R connection.
- d. You should leave SAS open when altering the configuration file.

12

Correct Answer: D

8.02 Poll – Correct Answer

The code below prints the first column of the data frame in the SAS Results Viewer.

- True
- False

```
call ExportDataSetToR("dog", "rmatrix");
submit;
  rmatrix[,1]
endsubmit;
```

21

Correct Answer: False

To send code to R, use the R option in the SUBMIT statement.

8.03 Poll – Correct Answer

Variable handles must be used in SAS Enterprise Miner.

- True
 False

59

Correct Answer: True

The object must be specified as &EMR_MODEL.

