



THE
POWER
TO KNOW.

SAS/IML[®] Studio 13.2

for SAS/STAT[®] Users

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2014. *SAS/IML® Studio 13.2 for SAS/STAT® Users*. Cary, NC: SAS Institute Inc.

SAS/IML® Studio 13.2 for SAS/STAT® Users

Copyright © 2014, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: 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.

For a Web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

August 2014

SAS provides a complete selection of books and electronic products to help customers use SAS® software to its fullest potential. For more information about our offerings, visit support.sas.com/bookstore or call 1-800-727-3228.

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.



Gain Greater Insight into Your SAS[®] Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.



Contents

Chapter 1.	Introduction to SAS/IML Studio	1
Chapter 2.	Reading and Writing Data	11
Chapter 3.	Creating Dynamically Linked Graphs	23
Chapter 4.	Calling SAS Procedures	27
Chapter 5.	Adding Variables to the DataObject	31
Chapter 6.	Adding Curves to Graphs	35
Chapter 7.	Reading ODS Tables	43
Chapter 8.	Adding Titles, Legends, and Insets	49
Chapter 9.	Adjusting Axes and Locations of Ticks	55
Chapter 10.	Changing the Color and Shape of Observation Markers	63
Chapter 11.	Calling Functions in the R Language	71

Index	79
--------------	-----------

Release Notes

The following release notes pertain to SAS/IML® Studio 13.2:

- For a complete list of new features in this release of SAS/IML Studio, see the SAS/IML Studio online Help: **Help ► Help Topics**. The following list highlights a few new features:
 - The IMLPlus language supports the experimental EXECUTEFILE subroutine, which runs IMLPlus statement in a text file. You can specify any text file that is accessible through the file system on the client.
 - The IMLPlus language now supports the EXECUTE subroutine.
- SAS/IML Studio includes documentation for Base SAS 9.4, SAS/IML 13.2, and SAS/STAT 13.2. See <http://support.sas.com/documentation/> for current versions of SAS documentation.
- If you need to open a data set that contains Chinese, Japanese, or Korean characters, it is important that you configure the “Regional and Language Options” in the Windows Control Panel for the appropriate country. It is not necessary to change the Windows setting called “Language for non-Unicode programs,” which is also referred to as the *system locale*.
- The SAS/IML Studio user interface is available in the following languages: English, Japanese, Korean, and Simplified Chinese. When you run SAS/IML Studio on a Windows system configured for a language other than English, you can still use English fonts by defining the *IMLStudio_ForceEnglishUI* environment variable. For more information, refer to the “Configuration” section of the chapter “The IML Studio Environment” in the SAS/IML Studio online Help.
- SAS/IML Studio uses the Microsoft Access Database Engine to import Microsoft Excel worksheets. The 32-bit edition of SAS/IML Studio requires the 32-bit edition of the Access Database Engine. Similarly, the 64-bit edition of SAS/IML Studio requires the 64-bit edition of the Access Database Engine.

Windows does not permit you to install the 32-bit and the 64-bit editions of Access Database Engine at the same time. If your Windows system does not have the appropriate edition of Access Database Engine installed, you will be unable to import Microsoft Excel worksheets by using the Open File dialog box or the IMLPlus method `DataObject.CreateFromExcelFile`. In this situation, you should use the IMPORT procedure, which is part of SAS/ACCESS® Interface to PC Files.

Chapter 1

Introduction to SAS/IML Studio

Contents

What Is SAS/IML Studio?	1
Purpose of This Book	3
Why Program in SAS/IML Studio?	3
Features of IMLPlus Programs	4
Understanding Classes, Objects, and Methods	5
The DataObject Class	6
Where Are the Data?	9

What Is SAS/IML Studio?

SAS/IML Studio (formerly known as SAS Stat Studio) is a programming environment for statistical computations and graphics. SAS/IML Studio requires that you have a license for Base SAS®, SAS/STAT®, and SAS/IML® software. SAS/IML Studio runs on a PC in the Microsoft Windows operating environment.

The programming language in SAS/IML Studio, which is called *IMLPlus*, is an enhanced version of the SAS/IML programming language. The “Plus” part of the name refers to new features that extend the SAS/IML language, including the ability to create and manipulate statistical graphs, to call SAS procedures, and to call functions in the R programming language.

SAS/IML Studio is intended for SAS programmers who want to implement algorithms that are not available in any SAS procedure, but SAS/IML Studio also provides a graphical user interface (GUI) to a number of standard statistical methods. You can use the SAS/IML Studio GUI to do the following:

- explore data through graphs linked across multiple windows
- transform data
- subset data
- analyze univariate distributions
- discover structure and features in multivariate data
- fit and evaluate explanatory models

Figure 1.1 shows the SAS/IML Studio interface with a logistic model for the probability that a passenger survived the 1912 *Titanic* disaster. The figure shows output from the LOGISTIC procedure and three linked views of the data: a data table, a diagnostic plot that uses the DIFCHISQ statistic to identify observations

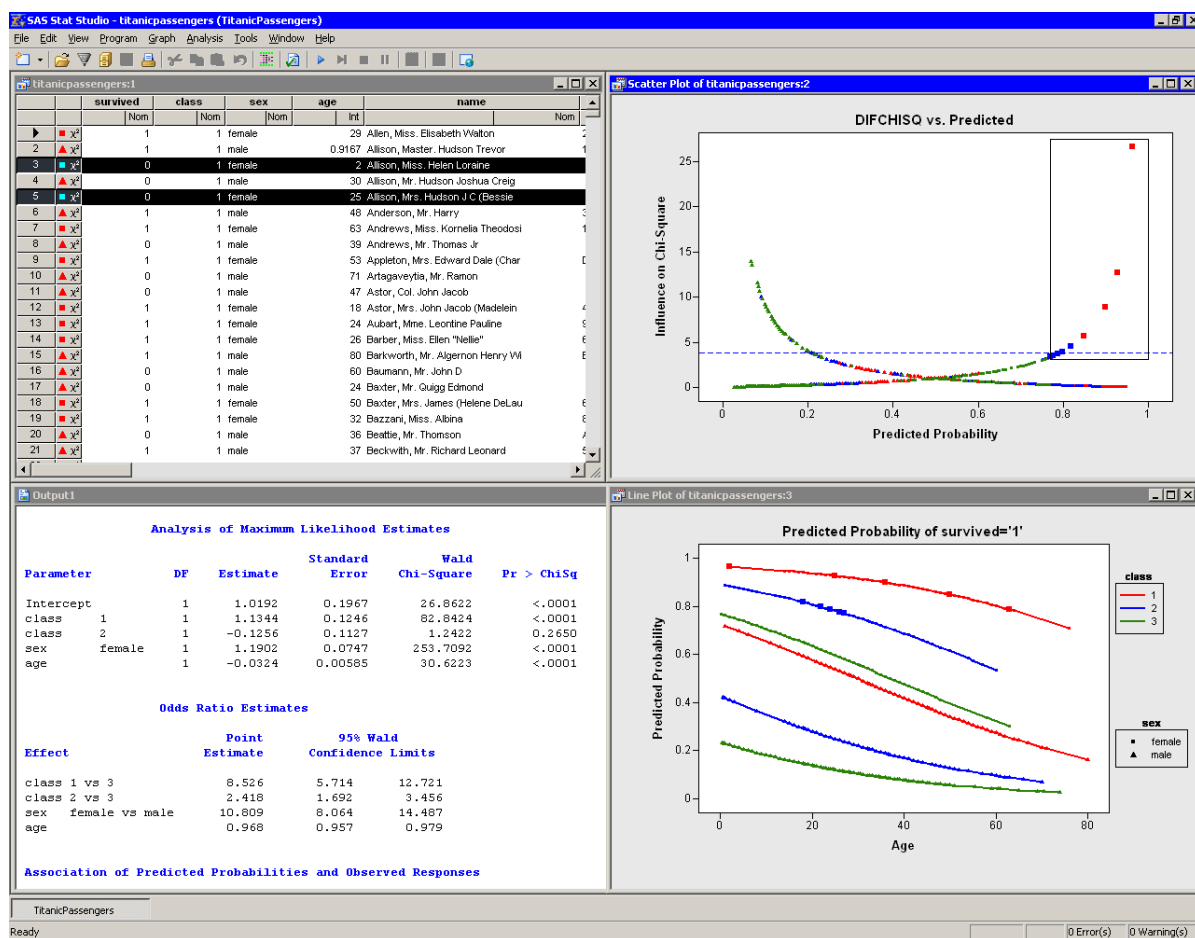
that do not fit the model well, and a line plot that shows the predicted probability of survival as a function of a passenger's age, gender, and cabin class (first class, second class, or third class). Observations that are selected in the diagnostic plot are shown as selected in all other (graphical and tabular) views of the data. The shapes and colors of observations are also shared among all views of the data.

Figure 1.1 was created by using only the SAS/IML Studio GUI. Although the GUI provides many tools for analyzing data, SAS/IML Studio is primarily intended for SAS programmers. SAS/IML Studio provides an integrated development environment that enables you to write, debug, and execute programs that combine the following:

- the flexibility of the SAS/IML matrix language
- the analytical power of SAS/STAT software
- the data manipulation capabilities of Base SAS software
- the dynamically linked graphs of SAS/IML Studio
- the functions and user-contributed packages of the R language

This book does not require previous knowledge of the SAS/IML language. The emphasis in this book is on the “Plus” part of the IMLPlus language.

Figure 1.1 The SAS/IML Studio Interface



Purpose of This Book

The purpose of this book is to teach SAS/STAT users how to use SAS/IML Studio in conjunction with SAS/STAT procedures in order to explore data and visualize statistical models. It assumes that you are familiar with using Base SAS software and SAS/STAT procedures such as FREQ, PRINT, REG, and KDE. The examples in this book do not require knowledge of the SAS/IML language. A goal of this book is to enable SAS/STAT programmers to write programs in SAS/IML Studio as quickly as possible.

In particular, this book focuses on how to create dynamically linked graphs so you can more easily formulate, visualize, evaluate, and revise statistical models. If you already know how to write DATA and PROC statements to perform a certain analysis, you can add a few IMLPlus statements to create graphs for visualizing the results. Thus, you need to learn only a few new commands and techniques in order to get started with IMLPlus programming.

This book is one of three documents about SAS/IML Studio. You can learn how to use the SAS/IML Studio GUI to conduct exploratory data analysis and standard statistical analyses in the *SAS/IML Studio User's Guide*. That book also shows you how to perform many of the tasks in this chapter by using menus in the SAS/IML Studio GUI. You can learn more advanced programming commands and techniques from the SAS/IML Studio online Help, which you can display by selecting **Help ► Help Topics** from the main menu.

Why Program in SAS/IML Studio?

Although you can use SAS/IML Studio as a point-and-click tool for exploratory data analysis, there are advantages to writing programs in SAS/IML Studio. Writing programs enables you to do the following:

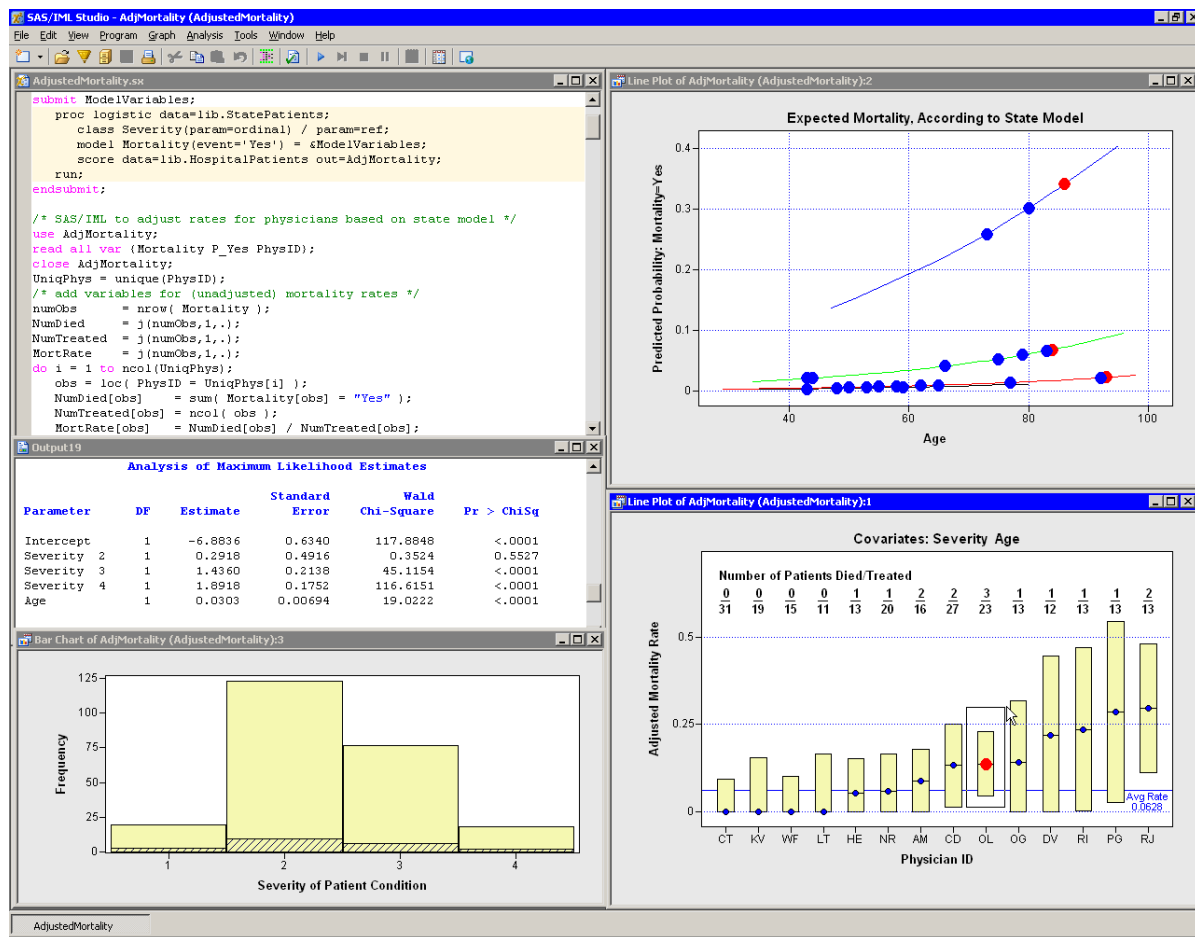
- create your own customized statistical graphs
- add legends, curves, maps, or other custom features to statistical graphs
- develop interactive programs that use dialog boxes
- extend the built-in analyses by calling SAS procedures
- create custom analyses
- repeat an analysis on different data
- extend the results of SAS procedures by using SAS/IML statements
- share analyses with colleagues who also use SAS/IML Studio
- call functions and user-contributed packages of the R language
- call functions from libraries written in C/C++, FORTRAN, or Java

Figure 1.2 shows the results of a program that evaluates the mortality of patients admitted to a certain hospital with congestive heart failure. The program uses a statewide database to build a logistic model that predicts mortality as a function of a patient's age and the severity of her condition. The program uses SAS/IML

statements to compute an adjusted mortality rate (with confidence limits) for cardiac physicians employed by the hospital. The adjusted rates are based on the observed number of deaths, the expected number of deaths (as predicted by the statewide model), and the mean number of deaths for this hospital.

The program implements many of the features listed previously. It creates a custom graphic with explanatory text. It calls DATA steps and the LOGISTIC procedure. It extends the results of the LOGISTIC procedure by using SAS/IML statements to compute adjusted mortality rates. It presents SAS/IML Studio's dynamically linked graphs to enable you to explore why some physicians have high rates of patient mortality, to decide whether those rates are unacceptably high, and to evaluate the overall performance of this hospital's staff compared to staff at other hospitals in the state. Although not shown in Figure 1.2, the program even uses a dialog box to enable you to choose the explanatory effects used to create the logistic model.

Figure 1.2 Results of an IMLplus Program



Features of IMLplus Programs

IMLplus programs such as the one that created Figure 1.2 share certain features. They typically include the following steps:

1. If the data are not already in a SAS data set in a library, put them there (Chapter 2).

2. Call a SAS procedure (Chapter 4).
3. Read in results produced by the procedure (Chapter 5, Chapter 6, and Chapter 7) and, optionally, use SAS/IML statements for additional analysis.
4. Create graphs that use the results (Chapter 3, Chapter 5, and Chapter 6).
5. Customize attributes of the graphs (Chapter 8, Chapter 9, and Chapter 10).

The chapters of this book describe these steps in detail. Later chapters build on earlier chapters. The last chapter (Chapter 11) discusses calling functions in the R language.

In each chapter, you write a short program that illustrates a few key ideas. You should type the program into the program window. You can create a new program window by selecting **File ► New Workspace** from the main SAS/IML Studio menu. For your convenience, the program for each chapter is also distributed with SAS/IML Studio.

The remainder of this chapter discusses concepts that are useful in IMLPlus programming but might not be familiar to the average SAS programmer.

- IMLPlus programs use *classes* and *methods* to manage dynamically linked graphs. The section “Understanding Classes, Objects, and Methods” on page 5 explains these concepts in general, and the section “The DataObject Class” on page 6 focuses on a class that is particularly important in SAS/IML Studio.
- IMLPlus programs that call SAS procedures require transferring data between an in-memory version of the data and SAS data sets. The section “Where Are the Data?” on page 9 introduces this concept. Chapter 2, “Reading and Writing Data,” discusses it in detail and gives examples.

Understanding Classes, Objects, and Methods

SAS is a procedural programming language. (So are FORTRAN and C.) SAS programming tends to be action-oriented. The procedure (or SAS/IML module) is the “unit” of programming. The procedure manipulates and analyzes the data.

In contrast, the IMLPlus programming language borrows ideas from object-oriented programming. An important idea in object-oriented programming is the concept of a *class*. A class is an abstract package of data and of functions (called *methods*) that query, retrieve, or manipulate the data.

An *object* is a concrete realization (or *instance*) of a class. To create an object, you need to specify the data to the creation routine for the class. To call methods in IMLPlus, you use a “dot notation” syntax in which the method name is appended to the name of the object. The form of the syntax is **Object.Method(arguments)**, as shown in the following examples.

In the SAS/IML language, all variables are matrices. In IMLPlus, a variable is implicitly assumed to be a SAS/IML matrix unless the variable is declared to refer to an object. You can specify that an IMLPlus variable refers to an object by using the **declare** keyword.

For example, to create a variable named **dobj** that refers to an object of the DataObject class, you can use the **CreateFromFile** method of the class:

```
declare DataObject dobj;  
dobj = DataObject.CreateFromFile("Hurricanes");  
dobj.Sort( "latitude" );
```

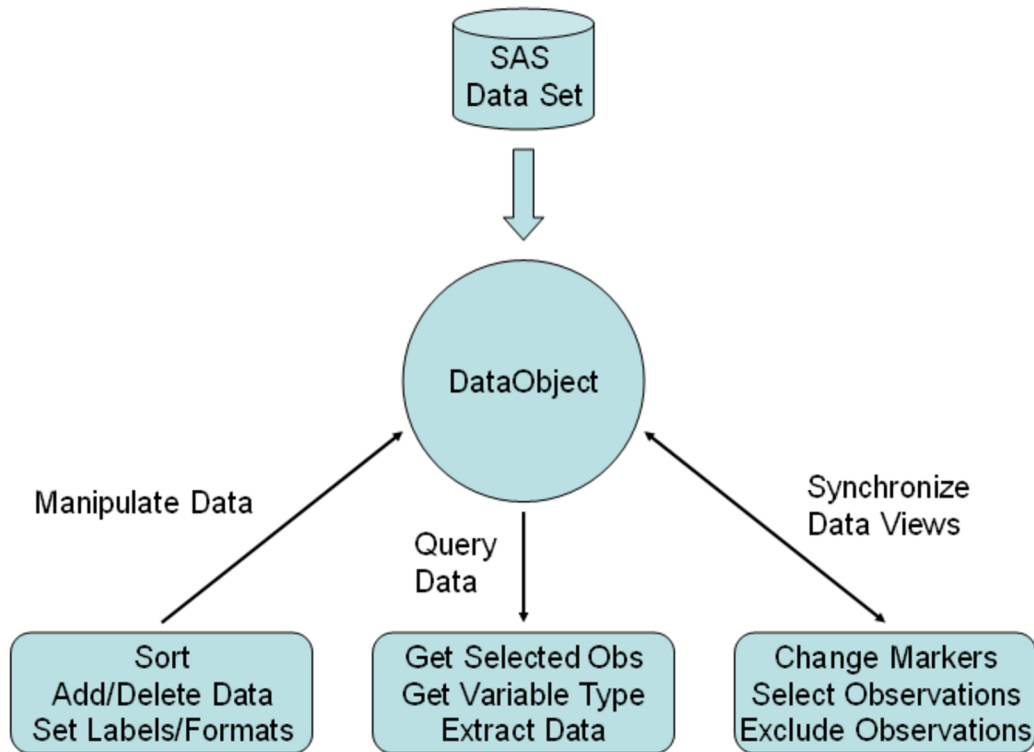
The `dobj` object is declared in the first line, created in the second, and manipulated in the third by calling a method. The `Sort` method sorts the data in `dobj` by the `latitude` variable. The data set on disk is not affected; it was used only to create the initial instance of the object.

To simplify the discussion, the remainder of this document refers to objects by the name of their class. Thus a `DataObject` object is called merely a “`DataObject`” instead of an “instance of the `DataObject` class.”

NOTE: SAS/IML statements are not case-sensitive. That is, if you define a matrix named **MyMatrix**, you can refer to the matrix as **mymatrix**, **MyMaTriX**, or any other combination of uppercase and lowercase letters. The names of IMLPlus classes and methods, however, *are* case-sensitive. There is no class named “`dataobject`” (lowercase), only “`DataObject`.” There is no method in the `DataObject` class named “`sort`,” only “`Sort`” (capitalized).

The DataObject Class

The most important class in SAS/IML Studio is the `DataObject` class. The `DataObject` class manages an *in-memory* version of your data. It provides methods to query, retrieve, and manipulate the data. It manages graphical information about observations such as the shape and color of markers, the selected state of observations, and whether observations are displayed in plots or hidden. [Figure 1.3](#) is a schematic depiction of a `DataObject`.

Figure 1.3 Using a DataObject

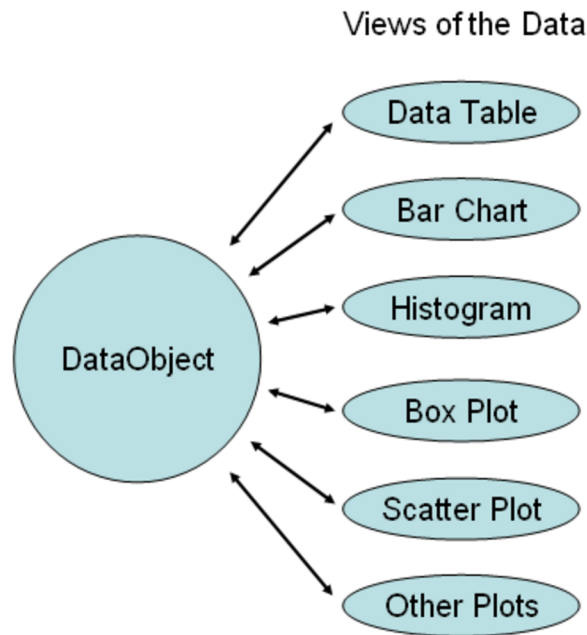
A DataObject is usually created from a SAS data set. (Other methods of creating DataObjects, such as from Microsoft Excel files or from SAS/IML matrices, are discussed in the online Help.) However, after the DataObject is created, the data in the DataObject are independent from the data used to initialize it. For example, you might use methods of the DataObject class to add new variables, transform existing variables, sort by one or more variables, delete observations, or exclude observations from being plotted. None of these operations affect the original SAS data set unless the DataObject is saved back onto disk with the same filename.

The DataObject class provides methods that query the data. For example, a DataObject can provide you with the number of variables and observations in the in-memory copy of the data. You can query for a variable's label or format, or for whether a variable contains nominal numeric data. You can request the DataObject to return a vector that contains the values of a particular variable. The values can then be used in a statistical analysis or to subset the data.

The DataObject class does not have any visible manifestation. Rather, you can create tabular and graphical views of the data from a DataObject. Every data table and every plot has an underlying DataObject, and usually several plots or tables share the same DataObject.

The most important role of the DataObject class is to synchronize all graphs and data tables that view the same data. Thus it is the DataObject class that enables dynamically linked views of data. This is schematically depicted in [Figure 1.4](#).

Figure 1.4 The DataObject Role



For example, the DataObject class keeps track of which observations are selected. When you interact with a graph or data table in order to select observations, your selections are remembered by the underlying DataObject. All graphs and tables that are linked to the DataObject are alerted so that they can update their displays to display the new set of selected observations.

Similarly, the DataObject class contains methods that manage markers for each observation. You can set the shape and color of an observation marker by using DataObject methods. Whenever an individual observation is plotted, it has the same shape and color in all graphs that display it.

In summary, the DataObject class is an in-memory version of data, together with methods to query and manipulate data and graphical attributes associated with observations. The purpose of the DataObject class is to ensure that all graphical and tabular views of the data display observations with the same markers and selection state.

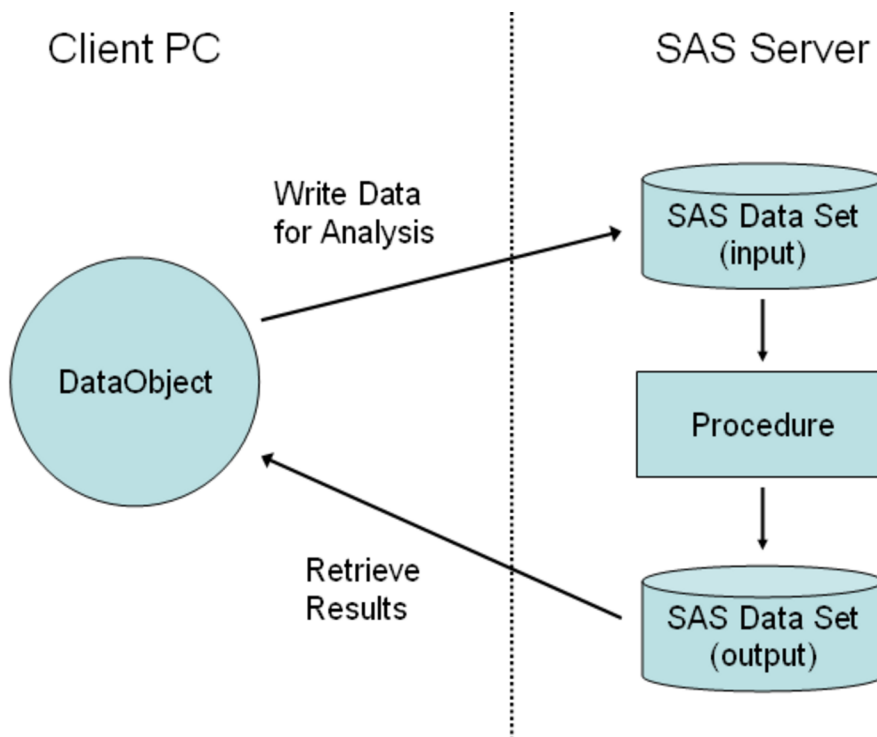
Where Are the Data?

SAS/IML Studio runs in a Microsoft Windows operating environment, but it can communicate with SAS software running on other computers. The PC on which SAS/IML Studio runs is called the *client*. The computer on which the SAS System runs is called the SAS *server*. If the SAS System is running on the same PC that is running SAS/IML Studio, then the client and server machines are the same.

There is a fundamental difference between the SAS/IML Studio graphs and the SAS/IML Studio analyses. The DataObject class, which coordinates all of the dynamically linked graphs and tables, runs on the client and keeps its data in memory on the client. Similarly, the graphs and tables run on the client. The analyses, by contrast, are performed using SAS procedures, and so the analyses run on the SAS server. The SAS procedures must read from a SAS data set in a library on the server.

To perform an analysis, you must get data out of the DataObject and write the data to a SAS data set in a server library. Similarly, after an analysis is complete, you might want to get the results (such as observation-wise statistics) out of a server data set and add them to the in-memory DataObject. Figure 1.5 illustrates this idea.

Figure 1.5 Data Flow



Thus it is important to know how to pass data between a DataObject and SAS data sets on the server. In Chapter 2, “[Reading and Writing Data](#),” you learn how to move variables between a DataObject and a server data set. You also learn how to read and write SAS data sets on the client or on the server, and how to create a DataObject from various sources of data.

Chapter 2

Reading and Writing Data

Contents

Introduction to Reading and Writing Data	11
Use the GUI to Read a SAS Data Set	12
Use the GUI to Write a SAS Data Set	15
Use a Program to Read a SAS Data Set	16
Use a Program to Write a SAS Data Set	18
Use SAS/IML Matrices to Store Data	19
Summary of Reading and Writing Data	21

Introduction to Reading and Writing Data

SAS/IML Studio runs in a Microsoft Windows operating environment, but it can communicate with SAS software that runs on other computers. The PC on which SAS/IML Studio runs is called the *client*. The computer on which the SAS system runs is called the SAS *server*. If the SAS System runs on the same PC as SAS/IML Studio, then the client and server machines are the same.

Dynamically linked graphs require an in-memory DataObject that runs on the client PC. Calling a SAS procedure requires a SAS data set in a library on the server. Therefore, if you are graphically exploring data and decide to perform an analysis with a procedure, you must write data from a DataObject into a SAS data set in a server library. After the analysis is finished, you might want to read results from an output data set and add one or more variables to the in-memory DataObject. For example, you might want to add predicted values, residuals, and confidence limits for a regression analysis.

This first part of this chapter teaches you how to use the SAS/IML Studio graphical user interface (GUI) to do the following:

- read a SAS data set from the client
- read a SAS data set from the server
- write data to a SAS data set on the client
- write data to a SAS data set on the server

The second part of the chapter teaches you how to do these tasks by writing a program and also describes how to add variables to an existing DataObject.

Use the GUI to Read a SAS Data Set

A SAS data set can be opened as a *client data set* if it is accessible by the PC operating system of the computer on which SAS/IML Studio runs. A data set on a USB flash drive, hard drive, CD drive, or DVD drive can be opened as a client data set. So, too, can a data set on a networked PC or a UNIX data set that is accessible through a mounted networked drive. For example, the following can be opened as client data sets:

- C:\Program Files\SAS\SASIMLStudio\12.1\Data Sets\Hurricanes.sas7bdat
- \\PC123\Public\Data\climate.sas7bdat
- U:\SAS Data\patients.sas7bdat

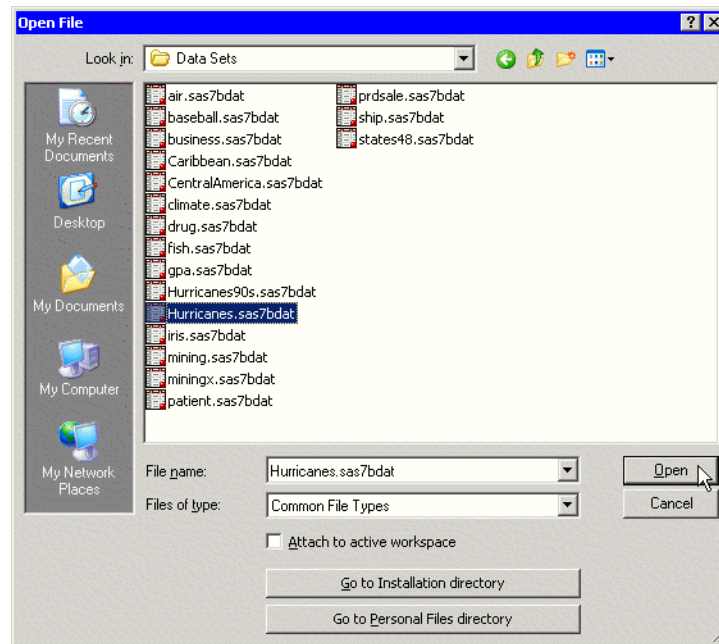
A SAS data set is a *server data set* if it is in a SAS library such as Work, Sasuser, or Sashelp, or in a libref that you define by using the LIBNAME statement. For example, the following are server data sets:

- Sashelp.Class
- Work.Data1
- MyLib.Research

Open a Client Data Set

To use the GUI to open a SAS data set on the client:

- 1 Select **File ► Open ► File** from the main menu. The dialog box in [Figure 2.1](#) appears.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Double-click the *Data Sets* folder.
- 4 Select the *Hurricanes.sas7bdat* file.
- 5 Click **Open**.

Figure 2.1 Opening a Client Data Set

A data table appears, showing a tabular view of the data. Connected to the data table (although invisible) is an underlying DataObject that was created from the SAS data set. The DataObject holds the data in memory; the data table displays a view of the data.

NOTE: Clicking **Go to Personal Files directory** navigates to your *personal files directory*. By default, the personal files directory corresponds to the Windows directory shown in [Table 2.1](#).

Table 2.1 The Personal Files Directory

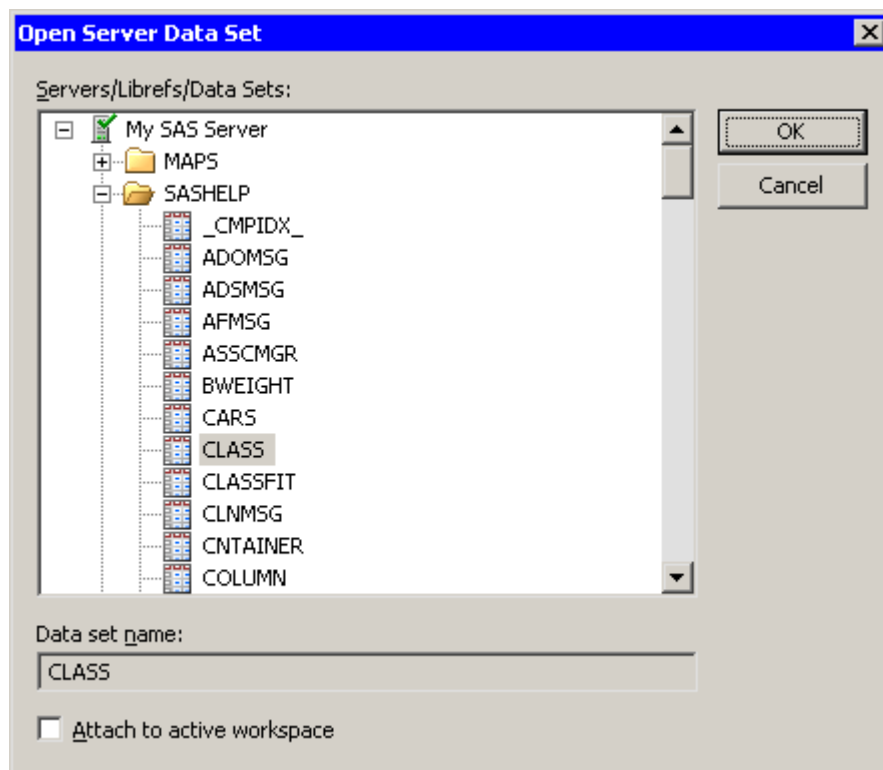
Windows XP	C:\Documents and Settings\ <i>userid</i> \My Documents\ My IML Studio Files
Windows Vista	C:\Users\ <i>userid</i> \Documents\My IML Studio Files
Windows 7	C:\Users\ <i>userid</i> \Documents\My IML Studio Files

Open a Server Data Set

To use the GUI to open a SAS data set in a library on the server:

- 1 Select **File ► Open ► Server Data Set** from the main menu. The dialog box in [Figure 2.2](#) appears.
- 2 Click the node labeled with your server name to open it. If the SAS System is running on your PC, the server name is **My SAS Server**.
- 3 Click the SASHELP folder to view the data sets in the Sashelp library.
- 4 Select the CLASS data set.
- 5 Click **OK**.

Figure 2.2 Opening a Server Data Set



A data table appears, showing a tabular view of the data. There is a DataObject (not visible, but still present) connected to the data table.

NOTE: [Figure 2.2](#) shows librefs that are not predefined. If your PC is your SAS server, you can create an *AutoExec.sas* file in the `C:\` root directory that contains LIBNAME statements that define librefs on your PC. Everytime a SAS server starts, the SAS System executes the *AutoExec.sas* file automatically. If you are running a SAS server on another computer, ask your site administrator to set up librefs for you.

Use the GUI to Write a SAS Data Set

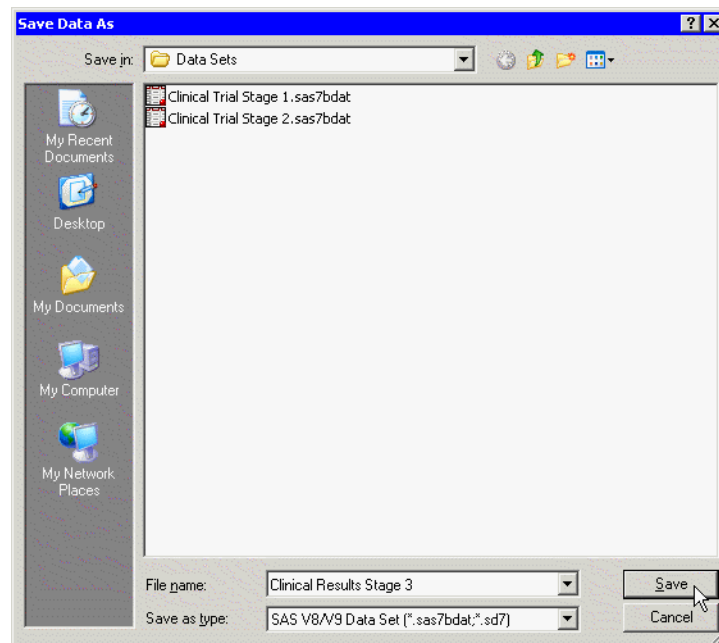
In addition to reading data sets, you can use the SAS/IML Studio GUI to write SAS data sets. This section presumes that you have opened a data table as described in the previous section.

Save Data to the Client

To use the GUI to save data from a data table (or more precisely, from the DataObject that underlies the data table) to a SAS data set on the client:

- 1 Activate the data table by clicking its title bar.
- 2 Select **File ► Save As File** from the main menu. The dialog box in [Figure 2.3](#) appears.
- 3 Navigate to the Windows directory in which you want to save the data set.
- 4 Type a valid Windows filename in the **File name** field.
- 5 Click **Save**.

Figure 2.3 Saving to a Client Data Set



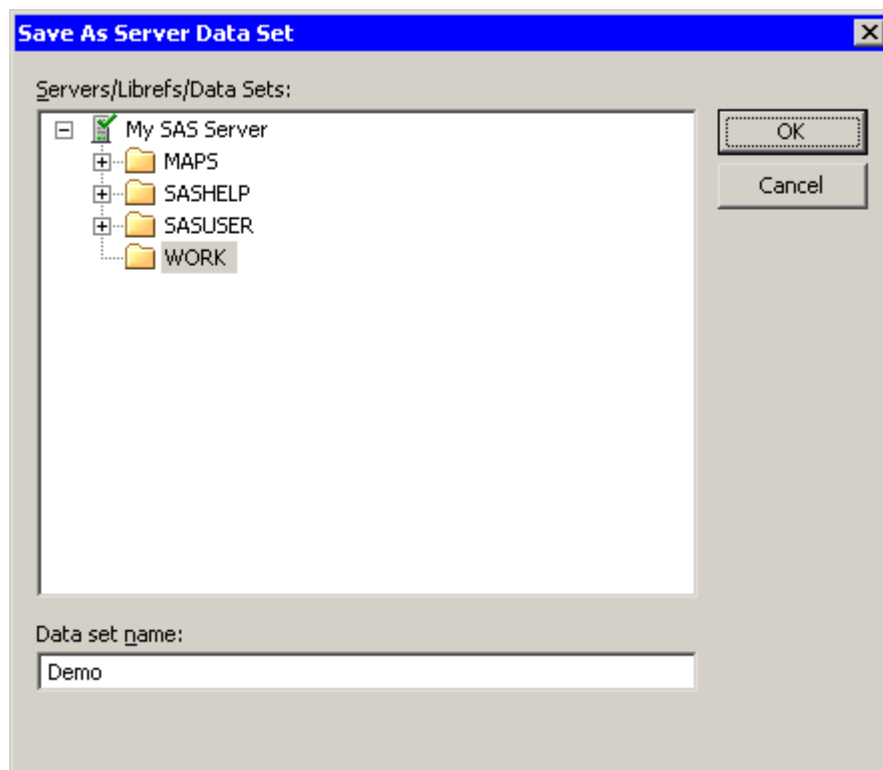
The recommended Windows directory in which to save your data is the personal files directory shown in [Table 2.1](#). If you have many data sets, you can organize the data by making subdirectories of this directory. SAS/IML Studio provides easy navigation for loading files in this directory. Furthermore, the section “[Open a Client Data Set](#)” on page 17 explains that this directory is automatically searched when you use a program to create a DataObject from a data set.

Save Data to a SAS Library

To use the GUI to save data from a data table (or more precisely, from the `DataObject` that underlies the data table) to a SAS data set in a library on the server:

- 1 Activate the data table by clicking its title bar.
- 2 Select **File ► Save As Server Data Set** from the main menu. The dialog box in [Figure 2.4](#) appears.
- 3 Click a node to see the available libraries for a server. If the SAS System is running on your PC, the server name is **My SAS Server**.
- 4 Click the node for a library. Each SAS/IML Studio workspace has its own private **Work** library, but other libraries (such as **Sasuser**) are shared across all SAS/IML Studio workspaces.
- 5 Type a valid SAS data set name in the **Data set name** field.
- 6 Click **OK**.

Figure 2.4 Saving to a SAS Library



Use a Program to Read a SAS Data Set

As explained in Chapter 1, “[Introduction to SAS/IML Studio](#),” the `DataObject` class stores an in-memory version of data. You can query, retrieve, and manipulate the data by calling methods in the `DataObject` class.

All graphical and tabular views of those data are linked together through the common `DataObject`. Therefore, when you want to create a graph of some data, or to look at the data in a table, you first need to create a `DataObject`.

A `DataObject` is typically created from a SAS data set. As explained in the introduction to this chapter, a client data set is accessible through the Windows operating environment, whereas a server data set resides in a SAS library.

In this section you write a program to create a `DataObject` from a SAS data set and save data from a `DataObject` to a SAS data set. You can open a program window by selecting **File ► New ► Workspace** from the main menu.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ► Open ► File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the `Data.sx` file.
- 5 Click **Open**.

Open a Client Data Set

To create a `DataObject` from a SAS data set on the client, you can use the `CreateFromFile` method of the `DataObject` class. Type or copy the following statements into a program window, and select **Program ► Run** from the main menu.

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");
DataTable.Create( dobj );
```

The first statement declares `dobj` to be an IMLPlus variable that refers to a `DataObject`. The second statement creates the `DataObject` and populates it with data from the specified data set. (More information about methods in the `DataObject` class is available in the SAS/IML Studio online Help.)

If you omit the file extension from the argument of `CreateFromFile` (as in this example), an extension of `sas7bdat` is assumed.

The last statement creates a `DataTable`, which displays the data in tabular form. The data table might appear behind your program window, so move the program window if necessary. While it is reassuring to see the data table and to know that your data set was correctly opened, this last statement is not necessary: the `DataObject` is created even if you do not create a `DataTable`.

NOTE: You can specify an absolute Windows pathname for the argument passed to the `CreateFromFile` method. However, if you specify a partial pathname as in the preceding example, then SAS/IML Studio searches for the file relative to certain directories. The `Hurricanes` data set is distributed with SAS/IML Studio. The directory that contains sample data sets (`C:\Program Files\SAS\SASIMLStudio\12.1\Data Sets`) is, by default, one of the directories automatically searched. See the SAS/IML Studio online Help for information about how to add or change directory search paths.

Open a Server Data Set

If your data are stored in a SAS data library, such as *Work*, *Sashelp*, or *Sasuser*, or in a libref that you created using the `LIBNAME` statement, you can open the data by using a similar method. The method's name is `CreateFromServerDataSet`. It is valid to have a `LIBNAME` statement in an IMLPlus program, so you can define your libref in the same program in which you use the `CreateFromServerDataSet` method.

Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
declare DataObject dobjServer;
dobjServer = DataObject.CreateFromServerDataSet("Sashelp.CLASS");
DataTable.Create( dobjServer );
```

Again, the data table might appear behind your program window, so move the program window if necessary. It is not necessary to create a `DataTable` unless you want to see a tabular view of the data.

If you already have a `DataObject` and you want to add variables from a server data set to it, then use the `CopyServerDataToDataObject` module as discussed in the section “[Use SAS/IML Matrices to Store Data](#)” on page 19 and Chapter 5, “[Adding Variables to the DataObject](#).”

Use a Program to Write a SAS Data Set

In the previous section you learned programming statements to read SAS data sets. In this section you learn programming statements to write SAS data sets.

Save Data to the Client

If you want to save data in a `DataObject` to a SAS data set on your PC, use the `WriteToFile` method.

Add the following statement at the bottom of the program window, and select **Program ► Run** from the main menu.

```
dobj.WriteToFile("C:\MyHurricanes.sas7bdat");
```

You must append the *sas7bdat* file extension to the filename when you use the `WriteToFile` method. If you omit the extension, you get an error message that notifies you that the file format is not supported.

It is often convenient to save data to your personal files directory, as described in the section “[Open a Client Data Set](#)” on page 12. You can get the Windows path for your personal files directory by using the `GetPersonalFilesDirectory` module. The following statements save data to the *Data Sets* folder under your personal files directory:

```
run GetPersonalFilesDirectory( path );
fname = path + "Data Sets\MyHurricanes.sas7bdat"; /* concatenate strings */
dobj.WriteToFile( fname );
```

Save Data to a SAS Library

If you want to save certain variables in a `DataObject` to a data set in a SAS library, use the `WriteVarsToServerDataSet` method of the `DataObject`.

Add the following statement at the bottom of the program window, and select **Program ► Run** from the main menu.

```
dobj.WriteVarsToServerDataSet( {"Name" "latitude" "longitude"},
    "Work", "HurrLatLon", true );
```

The first argument of the `WriteVarsToServerDataSet` method specifies the name of the variables in the `DataObject` that you want to write to the server. The next two arguments specify the SAS library (`Work`) and name of the data set (`HurrLatLon`) to be written. The last argument is a Java boolean argument. If you specify **true**, then observations that are marked “Exclude from Analysis” are not written to the server. If you specify **false**, then all observations are written to the server.

NOTE: Java is a case sensitive language, so the last argument of the `WriteVarsToServerDataSet` method must be lowercase.

If you want to write *all* variables to a SAS data set on the server, you can use the `WriteToServerDataSet` method. It is more efficient to write only the variables that are relevant to your analysis. For example, if you intend to run a regression that relates a single response variable to two explanatory variables, you should write only those three variables to the server data set.

View Data Sets in a SAS Library

There are two ways to verify that your data were written to a SAS library as you intended. The first way is to use the `PRINT` or `CONTENTS` procedure to print information to the output window. Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
submit;
proc print data=HurrLatLon(obs=10);
run;
endsubmit;
```

The `SUBMIT` and `ENDSUBMIT` statements are discussed in Chapter 4, “Calling SAS Procedures.” Anything between these two statements is passed from IMLPlus to the SAS System for processing. The result of these statements is to print the first 10 observations of the `Work.HurrLatLon` data set to the output window.

The second way to view a data set in a SAS library is to open the data set in SAS/IML Studio as explained in the section “Open a Server Data Set” on page 14.

Use SAS/IML Matrices to Store Data

Although this book does not require previous knowledge of the SAS/IML language, experienced SAS/IML programmers might wonder how to transfer data between matrices and a `DataObject`. For completeness, this issue is addressed in this section.

The SAS/IML language provides statements to read and write server data to and from matrices. The `USE` statement opens a SAS data set, and the `READ` statement reads server data into matrices. The SAS/IML language also has a `CREATE` statement that creates a server data set and an `APPEND` statement that writes variables from matrices. These statements are documented fully in the *SAS/IML User's Guide*.

Create a DataObject from SAS/IML Matrices

If you have data in SAS/IML matrices, you can create a `DataObject` from those matrices by using the `Create` method of the `DataObject`. For example, type the following statements into a SAS/IML Studio program window, and select **Program ► Run** from the main menu.

```
xy = {0 4,
      1 6,
      2 7,
      3 9,
      4 10};
declare DataObject dobjMatrix;
dobjMatrix = DataObject.Create("Matrix", {"XVar" "YVar"}, xy);
```

The 5×2 matrix `xy` is used to initialize the `dobjMatrix` `DataObject`. The result is a `DataObject` with two variables (named `XVar` and `YVar`) and five observations.

Add Variables to a DataObject

If the `DataObject` is already created, you can add a new variable to the `DataObject` by using the `AddVar` method. For example, add the following statements to the SAS/IML Studio program from the previous section, and select **Program ► Run** from the main menu.

```
z = {-1, 1, 0, 0, 1};
dobjMatrix.AddVar( "z", z);
DataTable.Create( dobjMatrix );
```

The result is a `DataObject` with three variables (the new variable is `z`) and five observations. It is an error to try to add a new variable that has a different number of observations than the `DataObject`.

There is also a `DataObject` method, called the `AddVars` method, that enables you to add several variables in a single call.

Copy Variables to SAS/IML Matrices

You can copy data from the `DataObject` into a matrix by using the `GetVarData` method of the `DataObject`. For example, add the following statements to the SAS/IML Studio program from the previous section, and select **Program ► Run** from the main menu.

```
dobjMatrix.GetVarData( "XVar", x);
print x;
```

The result is a 5×1 matrix `x` that contains the values of the `XVar` variable. You can name the matrix anything you want. It can have the same name as the variable in the `DataObject` (as the `z` variable and matrix did) or a completely different name (as in this example). After the data are copied into a matrix, there is no linkage

between the DataObject and the matrix: changing elements of the matrix does not affect the DataObject, and changing the DataObject does not affect the matrix.

The CopyServerDataToDataObject module uses this approach to read variables in a server data set directly into a DataObject. This module is discussed in Chapter 5, “[Adding Variables to the DataObject](#).” Again, the number of observations in the server data set must match the number of observations in the DataObject.

Summary of Reading and Writing Data

Tables 2.2 through 2.4 summarize reading and writing data in IMLPlus programs.

Table 2.2 Reading Data into a DataObject

Source	Statement
Client data set	DataObject.CreateFromFile
Server data set	DataObject.CreateFromServerDataSet run CopyServerDataToDataObject module
SAS/IML matrix	DataObject.Create DataObject.AddVar DataObject.AddVars

Table 2.3 Copying Data from a DataObject

Destination	Statement
Client data set	DataObject.WriteToFile
Server data set	DataObject.WriteToServerDataSet DataObject.WriteVarsToServerDataSet
SAS/IML matrix	DataObject.GetVarData

Table 2.4 Reading and Writing Server Data in SAS/IML

Action	Statement
Read from server into matrices	USE, READ
Write server data set from matrices	CREATE, APPEND

NOTE: If you create a server data set by using the WriteToServerDataSet or WriteVarsToServerDataSet method, you might want to delete the data set after you are finished with it by calling the DELETE subroutine.

Chapter 3

Creating Dynamically Linked Graphs

Contents

Introduction to Creating Dynamically Linked Graphs	23
Create a Scatter Plot and Histogram	23
Select Observations	24
Summary of Creating Graphs	26

Introduction to Creating Dynamically Linked Graphs

Dynamically linked graphs are one of the primary tools of exploratory data analysis. When you have multivariate data and you want to understand relationships between variables, it is often useful to create dynamically linked graphs.

Chapter 2, “[Reading and Writing Data](#),” demonstrated ways to open SAS data sets stored on your PC or on a SAS server. In this chapter you create standard statistical graphs to visualize your data and relationships between variables.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ►Open ►File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the *Graphs.sx* file.
- 5 Click **Open**.

Create a Scatter Plot and Histogram

In this example, you write a program to create a scatter plot and a histogram from variables in a DataObject. Type or copy the following statements into a program window, and select **Program ►Run** from the main menu.

```

declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");

declare ScatterPlot plot;
plot = ScatterPlot.Create( dobj, "min_pressure", "wind_kts" );

declare Histogram hist;
hist = Histogram.Create( dobj, "latitude" );
hist.SetWindowPosition( 50, 50, 50, 50 );

```

The first set of statements create a `DataObject` and are explained in Chapter 2, “[Reading and Writing Data](#).” The next statements declare `plot` to be an IMLPlus variable that refers to a `ScatterPlot`, which is the class that displays and manipulates attributes of a scatter plot. The scatter plot in this example is created from the `min_pressure` and `wind_kts` variables in the `DataObject`. The `min_pressure` variable contains the minimum central pressure in hectopascals (1 hPa = 1 millibar), and the `wind_kts` variable contains the maximum wind speed in knots for each observation of a tropical cyclone.

The final set of statements declare `hist` to be a `Histogram`, which is the class that displays and manipulates attributes of a histogram. The histogram is created from the `latitude` variable. The `latitude` variable contains the latitude in degree (north of the equator) at which the observation was made.

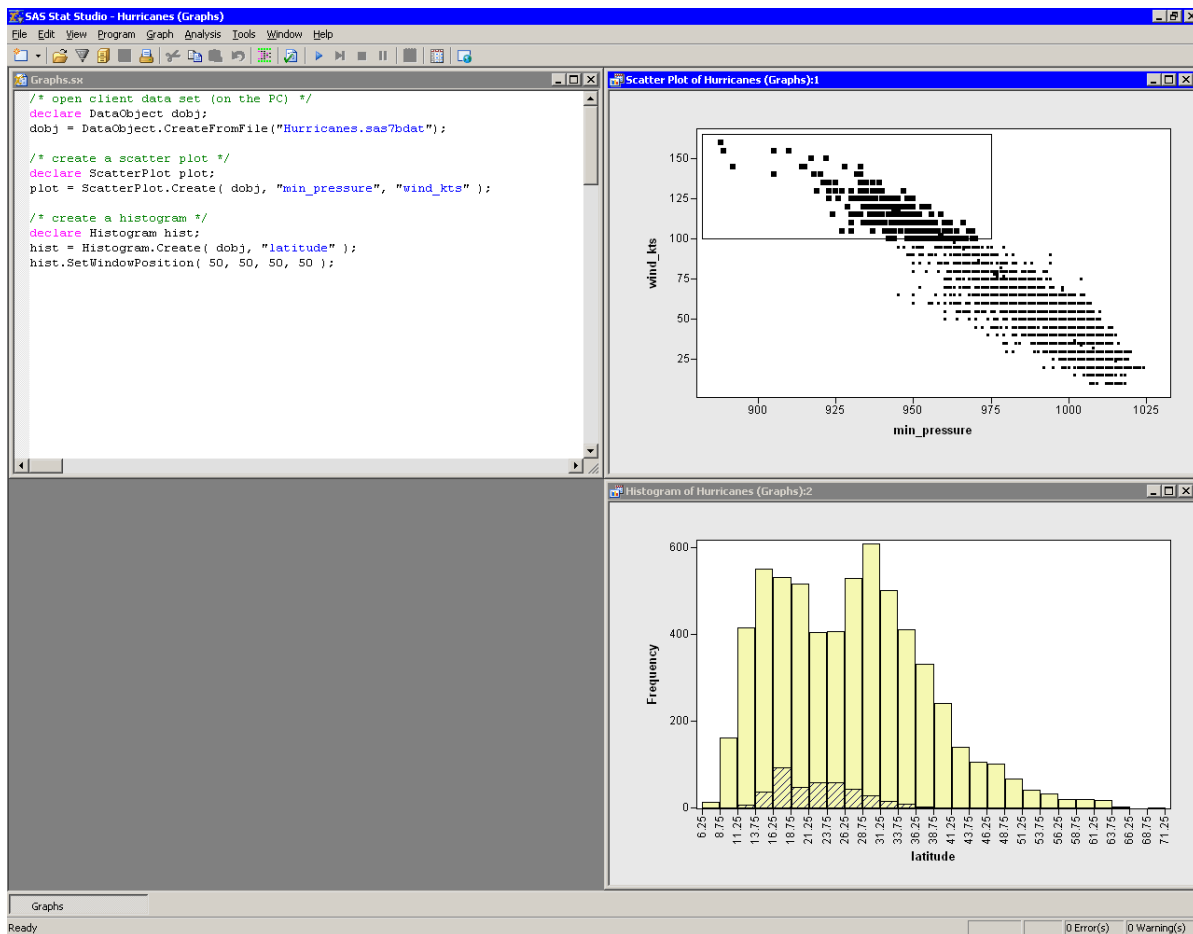
To avoid having the histogram appear on top of the scatter plot, the `SetWindowPosition` method is called to move the histogram into the lower right corner of the SAS/IML Studio workspace. The `SetWindowPosition` method is a method in the `DataView` class. Any plot or data table can call methods in the `DataView` class. (Formally, the `DataView` class is a *base class* for all plots and data tables.) The `DataView` class is documented in the SAS/IML Studio online Help.

Select Observations

You can now select observations in either graph and see how the latitude of Atlantic cyclones is related to the wind speed and minimum pressure. You can select observations in a plot by clicking on observations or on bars. You can add to a set of selected observations by holding the CTRL key and clicking. You can also select observations by using a *selection rectangle*. To create a selection rectangle, click in a graph and hold down the left mouse button while you move the mouse pointer to a new location.

In the scatter plot, use a selection rectangle to select all observations with wind speed greater than or equal to 100 knots. The result is shown in [Figure 3.1](#).

Figure 3.1 Creating Linked Graphs

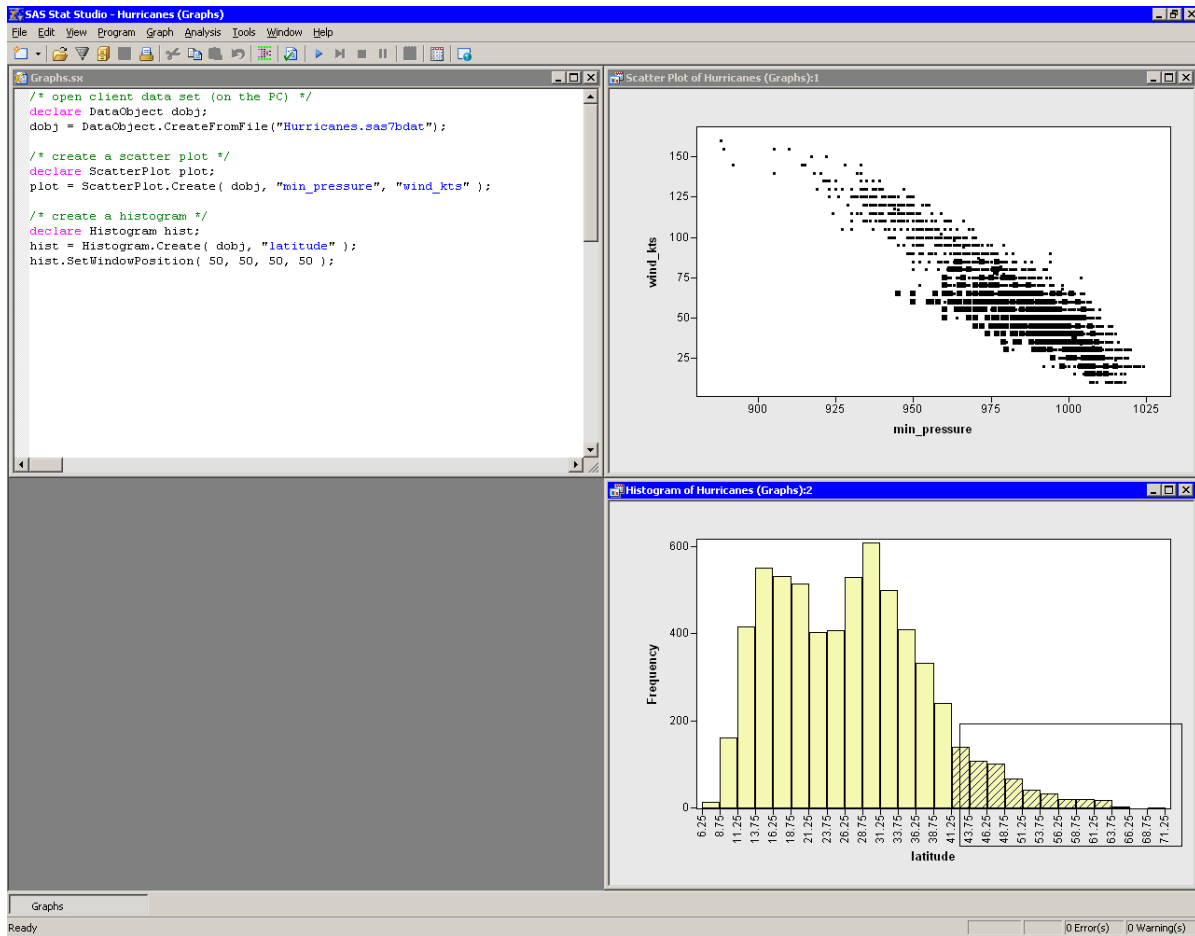


Note that these very intense storms tend to occur in southern latitudes, primarily between 14 and 35 degrees of latitude.

In the histogram, use a selection rectangle to select all observations with latitude greater than 41 degrees. The result is shown in [Figure 3.2](#).

The National Hurricane Center classifies a cyclone as a hurricane if it has sustained low-level winds of 64 knots or greater. Note that few of the storms that enter northern latitudes are still strong enough to be classified as hurricanes. Northern storms tend to have relatively low wind speeds.

Figure 3.2 Storms in Relatively Northern Latitudes



Summary of Creating Graphs

In this chapter you learned how to use IMLPlus statements to create a scatter plot and a histogram. Many other statistical graphs are available to you in SAS/IML Studio. Consult the SAS/IML Studio online Help for documentation on all classes of statistical graphs available in SAS/IML Studio.

Chapter 4

Calling SAS Procedures

Contents

Introduction to Calling a SAS Procedure	27
Create a Data Set for the Procedure	27
Call a Procedure	28
Substitute Values from SAS/IML Matrices	29

Introduction to Calling a SAS Procedure

In previous chapters you learned how to open a data set and how to explore the data by creating dynamically linked graphs. Qualitatively exploring your data might lead you to formulate statistical models. You can use Base SAS and SAS/STAT procedures to quantitatively analyze your data.

This chapter shows you how to call SAS procedures from an IMLPlus program. The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ► Open ► File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the *Proc.sx* file.
- 5 Click **Open**.

Create a Data Set for the Procedure

Recall from Chapter 1, “Introduction to SAS/IML Studio,” that the DataObject, which coordinates all of the dynamically linked graphs and tables, runs on the client and keeps its data in memory on the client. However, SAS procedures run on the SAS server and read data from a SAS data set in a library. Therefore, to perform an analysis, you must get data out of the DataObject and write the data to a SAS data set in a server library.

In Chapter 3, “Creating Dynamically Linked Graphs,” you created a scatter plot of `wind_kts` versus `min_pressure` for the Hurricanes data. You might have noticed that the scatter plot reveals a linear relationship between these two variables. In this section you call the REG procedure to fit a linear least squares model.

Type or copy the following statements into a program window.

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");

dobj.WriteVarsToServerDataSet( {"wind_kts" "min_pressure"},
    "Work", "Hurr", true );
```

These statements create a DataObject from the Hurricanes data set on your PC and write the wind_kts and min_pressure variables to a server data set called Work.Hurr. These statements are explained in Chapter 2, “Reading and Writing Data.”

NOTE: If your data are already in a data library on the SAS server (for example, Sasuser), then the previous statements are not necessary. You can just reference the data set in the DATA= option of the procedure.

Call a Procedure

Now that a SAS data set that contains variables of interest is on the server, you can call any SAS procedure to analyze the data. In order to tell the IMLPlus language that you want certain statements to be sent to the SAS System rather than interpreted as IMLPlus, you must bracket your SAS statements with SUBMIT and ENDSUBMIT statements.

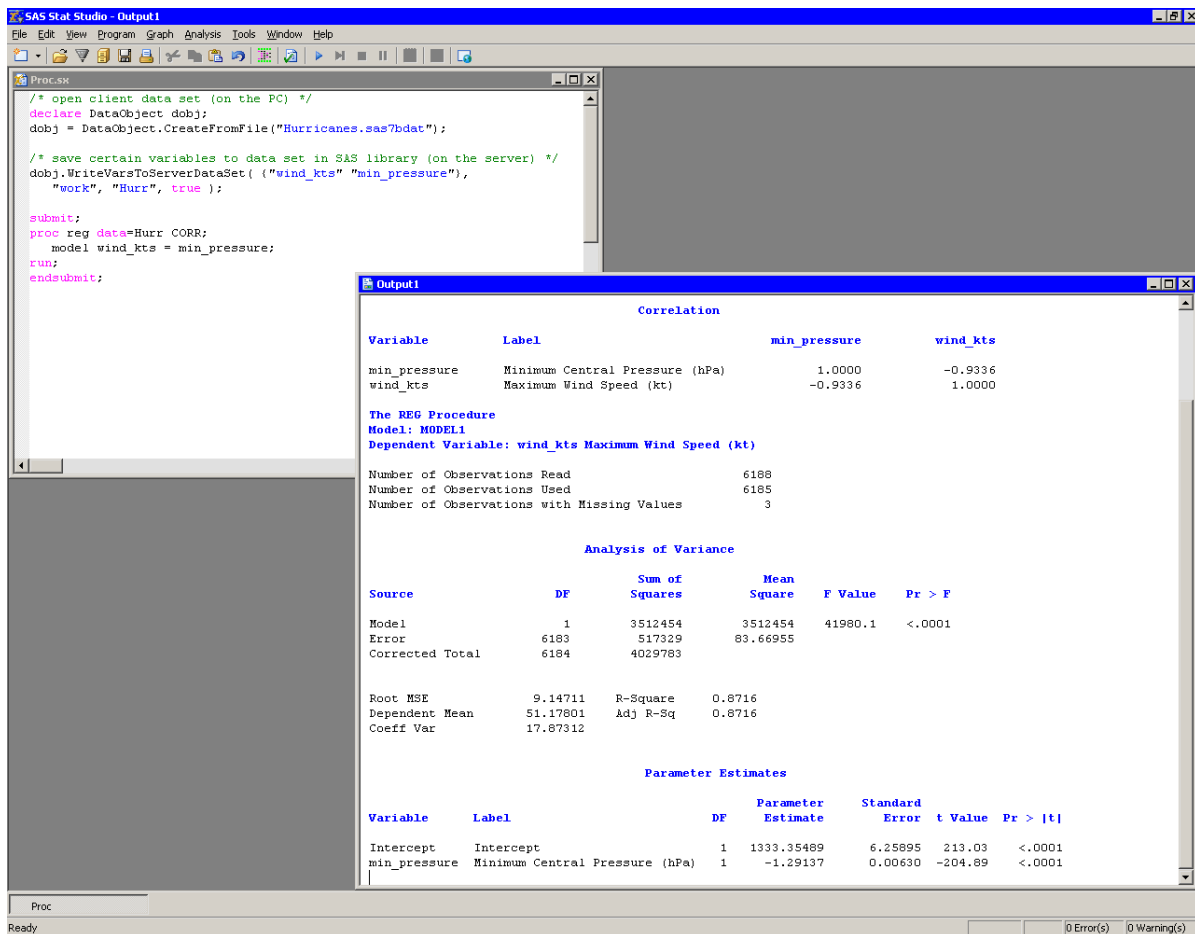
Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
submit;
proc reg data=Hurr CORR;
    model wind_kts = min_pressure;
run;
endsubmit;
```

When you run the program, SAS/IML Studio calls the REG procedure with the CORR option. The CORR option prints out the correlation matrix for the variables in the MODEL statement. (If you are using the HTML output destination, and if ODS graphics are on, you will also see plots that are created by the REG procedure.)

The REG procedure displays its tables in the SAS/IML Studio output window as shown in [Figure 4.1](#). From the output you can see that the correlation between these two variables is -0.9336 . The R-square value is 0.8716 for the line of least squares given approximately by $\text{wind_kts} = 1333 - 1.291 \times \text{min_pressure}$.

Figure 4.1 Calling the REG Procedure



Substitute Values from SAS/IML Matrices

The SUBMIT statement enables you to substitute the values of a SAS/IML matrix into the statements that are submitted to the SAS System. For example, the following program submits SAS statements that are identical to the preceding example:

```

YVar = "wind_kts";
XVar = "min_pressure";
submit XVar YVar;
proc reg data=Hurr CORR plots=none;
  model &YVar = &XVar;
run;
endsubmit;

```

You can list the names of SAS/IML matrices on the SUBMIT statement and refer to the contents of those matrices inside the SUBMIT/ENDSUBMIT block. The syntax is reminiscent of the syntax for macros: an ampersand (&) preceding an expression means “substitute the value of the expression.” However, the substitution takes place before the SUBMIT/ENDSUBMIT block is sent to the SAS System; no macro variables are actually created. You can substitute character values, as shown in this example, or numeric values. If **x** is a vector, then **&x** lists the elements of **x** separated by spaces.

Chapter 5

Adding Variables to the DataObject

Contents

Introduction to Adding Variables to a DataObject	31
Add a Variable to a DataObject	31

Introduction to Adding Variables to a DataObject

In previous chapters, you learned how to open a data set and how to call a SAS procedure by using the SUBMIT and ENDSUBMIT statements. This chapter shows you how to read observation-wise statistics from the output data set of a procedure, and how to add these variables to the DataObject so that you can visualize the results.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ► Open ► File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the `AddVar.sx` file.
- 5 Click **Open**.

Add a Variable to a DataObject

Type or copy the following statements into a program window, and select **Program ► Run** from the main menu.

```
declare DataObject dobj;  
dobj = DataObject.CreateFromFile("Hurricanes");  
  
dobj.WriteVarsToServerDataSet( {"wind_kts" "min_pressure"},  
    "Work", "Hurr", true );
```

These statements open the Hurricanes data set from your PC and write the `wind_kts` and `min_pressure` variables to a server data set called `Work.Hurr`. These statements are explained in Chapter 2, “[Reading and Writing Data](#).”

In the Chapter 4, “[Calling SAS Procedures](#),” you called the `REG` procedure with the `Work.Hurr` data and viewed tables and statistics in the output window. This time, you use the `OUTPUT` statement to create an output data set that includes the residual values for the regression model.

Add the following statements at the bottom of the program window, and select **Program ►Run** from the main menu.

```
submit;
proc reg data=Hurr plots=none;
    model wind_kts = min_pressure;
    output out=RegOut R=Residual;
run;
endsubmit;
```

When you run the program, SAS/IML Studio calls the `REG` procedure. The procedure creates an output data set named `Work.RegOut` that contains all of the original variables in `Work.Hurr`, plus a new variable named `Residual`. This variable is created by the `R=` option in the `OUTPUT` statement.

Now that an output variable is created, you can add it to the `DataObject`. You can read variables in a server data set directly into a `DataObject` by using the `CopyServerDataToDataObject` module. (Note that the number of observations in `Work.RegOut` matches the number of observations in the `dobj` `DataObject`.) After a variable is in the `DataObject`, you can use that variable to create graphs that help to visualize the analysis. In this case, you can create a plot of the residuals versus the explanatory variable.

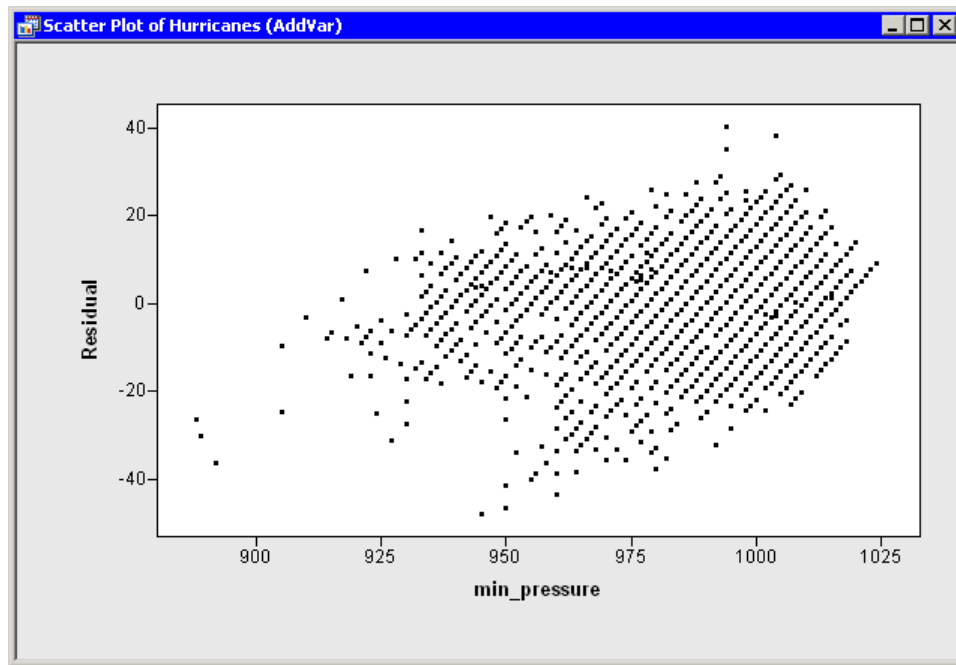
Add the following statements at the bottom of the program window, and select **Program ►Run** from the main menu.

```
ok = CopyServerDataToDataObject( "Work", "RegOut", dobj,
    {"Residual"}, /* name on server */
    {"Residual"}, /* name in DataObject */
    {"Residuals"}, /* label in DataObject */
    true /* if an existing variable has this name, replace it */
);

declare ScatterPlot ResPlot;
ResPlot = ScatterPlot.Create( dobj, "min_pressure", "Residual" );
```

The residual plot ([Figure 5.1](#)) shows many storms with large negative residuals. These storms had much lower wind speeds (20–40 knots lower) than predicted from their values of `min_pressure`.

Figure 5.1 Creating a Residual Plot



Chapter 6

Adding Curves to Graphs

Contents

Introduction to Adding Curves to Graphs	35
Call a Procedure to Create Data for a Curve	35
Draw a Reference Line	36
Draw a Curve from Variables in the DataObject	37
Draw a Curve from Variables in a Data Set or from a SAS/IML Matrix	41

Introduction to Adding Curves to Graphs

In previous chapters you learned how to open a data set and how to call a SAS procedure by using the SUBMIT and ENDSUBMIT statements. You learned how to read output variable from the server into the DataObject. This chapter shows you how to add lines and curves to a graph to visualize a model's fit.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ►Open ►File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the *Fit.sx* file.
- 5 Click **Open**.

Call a Procedure to Create Data for a Curve

In Chapter 5, “Adding Variables to the DataObject,” you created a residual plot. In this chapter you also create a scatter plot of the independent and dependent variables that shows the model's predicted values and the upper and lower 95% limits for individual predictions. The REG procedure outputs the predicted values with the P= option in the MODEL statement, and the upper and lower prediction limits with the LCL= and UCL= options.

Type or copy the following statements into a program window, and select **Program ►Run** from the main menu.

```

declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");

dobj.WriteVarsToServerDataSet( {"wind_kts" "min_pressure"},
    "Work", "Hurr", true );

submit;
proc reg data=Hurr plots=none;
    model wind_kts = min_pressure;
    output out=RegOut P=Pred R=Residual LCL=LCL UCL=UCL;
run;
endsubmit;

ok = CopyServerDataToDataObject( "Work", "RegOut", dobj,
    {"Pred" "Residual" "LCL" "UCL"}, /* names on server */
    {"Pred" "Residual" "LCL" "UCL"}, /* names in DataObject */
    {"Predicted" "Residuals" "Lower Conf. Limit" "Upper Conf. Limit"},
    true );

```

The first two sets of statements are explained in Chapter 2, “[Reading and Writing Data](#).” The next two sets of statements are similar to statements in the example from Chapter 5, “[Adding Variables to the DataObject](#).”

Whenever you want to add a line or curve to a plot, you need to specify the following:

- whether the curve will be drawn on top of observations or under them
- the coordinate system in which the curve will be drawn
- the color, line style, and line width of the curve
- the coordinates of the curve

These aspects are discussed in the SAS/IML Studio online Help.

Draw a Reference Line

You can create a residual plot of the residuals versus the explanatory variable as before. However, this time you add a horizontal reference line to the plot to indicate where residuals are zero.

By default, curves are drawn on top of observations, which is fine for this example. The following statements draw a dashed black line at Residual = 0.

Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```

declare ScatterPlot ResPlot;
ResPlot = ScatterPlot.Create( dobj, "min_pressure", "Residual" );
ResPlot.DrawUseDataCoordinates();
ResPlot.DrawSetPenStyle( DASHED );
ResPlot.DrawLine( 850, 0, 1150, 0 ); /* from (850,0) to (1150,0) */

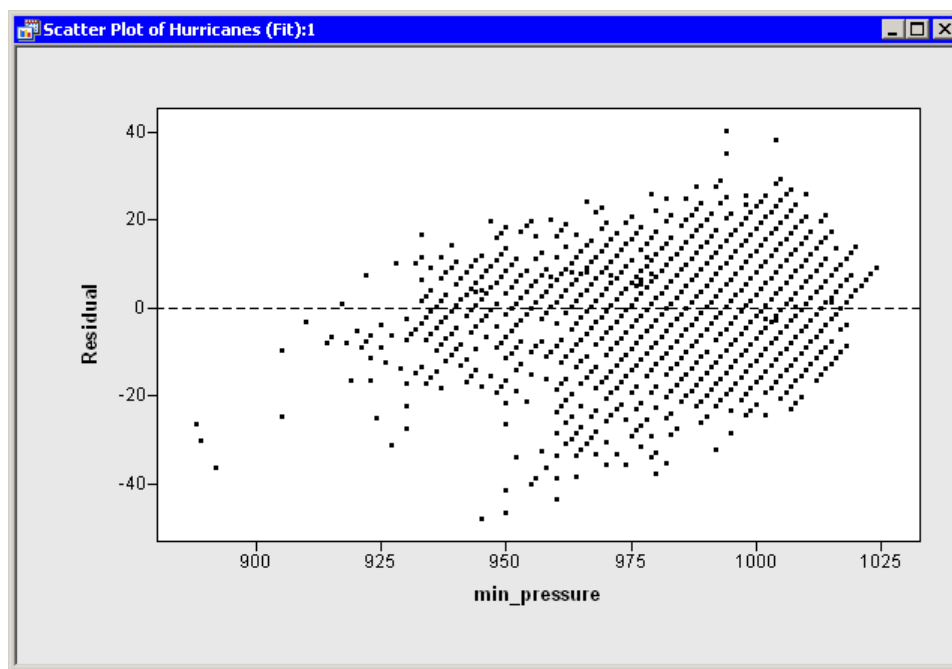
```

The `DrawUseDataCoordinates` method is a method in the `Plot` class. The `ScatterPlot` class can use all the methods in the `Plot` class. (Formally, the `Plot` class is a *base class* for the `ScatterPlot` class.) The `DrawUseDataCoordinates` method specifies that the coordinate system for the reference line is the same as the coordinate system for the plot's data.

If you do not specify otherwise, a curve in a plot is drawn as a solid black line with a one-pixel width. The `DrawSetPenStyle` method specifies that you want to override the default line style (`SOLID`) with a different style (`DASHED`).

The `DrawLine` method specifies the two endpoints of the line segment. In this example, a line is drawn on the plot from (850, 0) to (1150, 0). The line extends across the entire plot area since both endpoints are beyond the range of the `min_pressure` variable. Figure 6.1 shows the resulting plot.

Figure 6.1 A Plot with a Reference Line



Draw a Curve from Variables in the DataObject

Now you will create a scatter plot of `wind_kts` versus `min_pressure`. On this plot you add the predicted values and the upper and lower confidence limits as computed by the REG procedure. This fitted model is a straight line, which can be visualized by specifying two points as you did for the residual plot. However, visualizing a more general parametric model (for example, a quadratic model in `min_pressure`) or a nonparametric model requires that you plot a *polyline*, so this example adopts the general approach.

A *polyline* is a sequence of connected line segments specified by passing two vectors to the `DrawLine` method. The first vector specifies the X coordinates, and the second specifies the Y coordinates. The polyline is drawn from (x_1, y_1) to (x_2, y_2) to (x_3, y_3) and so on until (x_n, y_n) .

To draw the polyline, you need a SAS/IML matrix that contains the coordinates to be plotted. You can use the `GetVarData` method of the `DataObject` class to get the data from the `DataObject`. You also need to sort the

data according to the X variable, which is `min_pressure` for this example. (Alternatively, you can use the SAS/IML SORT call if you do not want to change the order of observations in the DataObject.)

Add the following statements at the bottom of the program window, and select **Program ►Run** from the main menu.

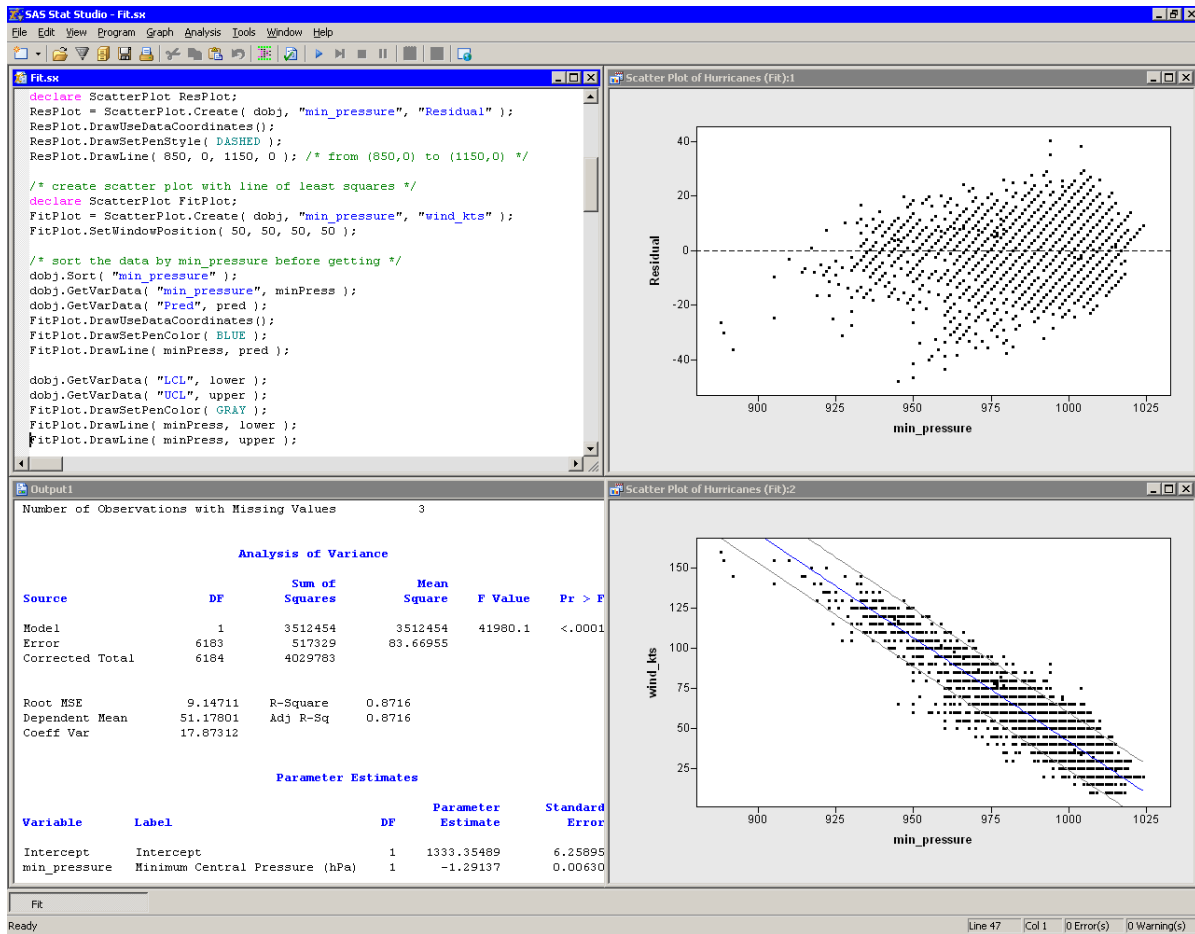
```
declare ScatterPlot FitPlot;
FitPlot = ScatterPlot.Create( dobj, "min_pressure", "wind_kts" );
FitPlot.SetWindowPosition( 50, 50, 50, 50 );

dobj.Sort( "min_pressure" );
dobj.GetVarData( "min_pressure", minPress );
dobj.GetVarData( "Pred", pred );
FitPlot.DrawUseDataCoordinates();
FitPlot.DrawSetPenColor( BLUE );
FitPlot.DrawLine( minPress, pred );

dobj.GetVarData( "LCL", lower );
dobj.GetVarData( "UCL", upper );
FitPlot.DrawSetPenColor( GRAY );
FitPlot.DrawLine( minPress, lower );
FitPlot.DrawLine( minPress, upper );
```

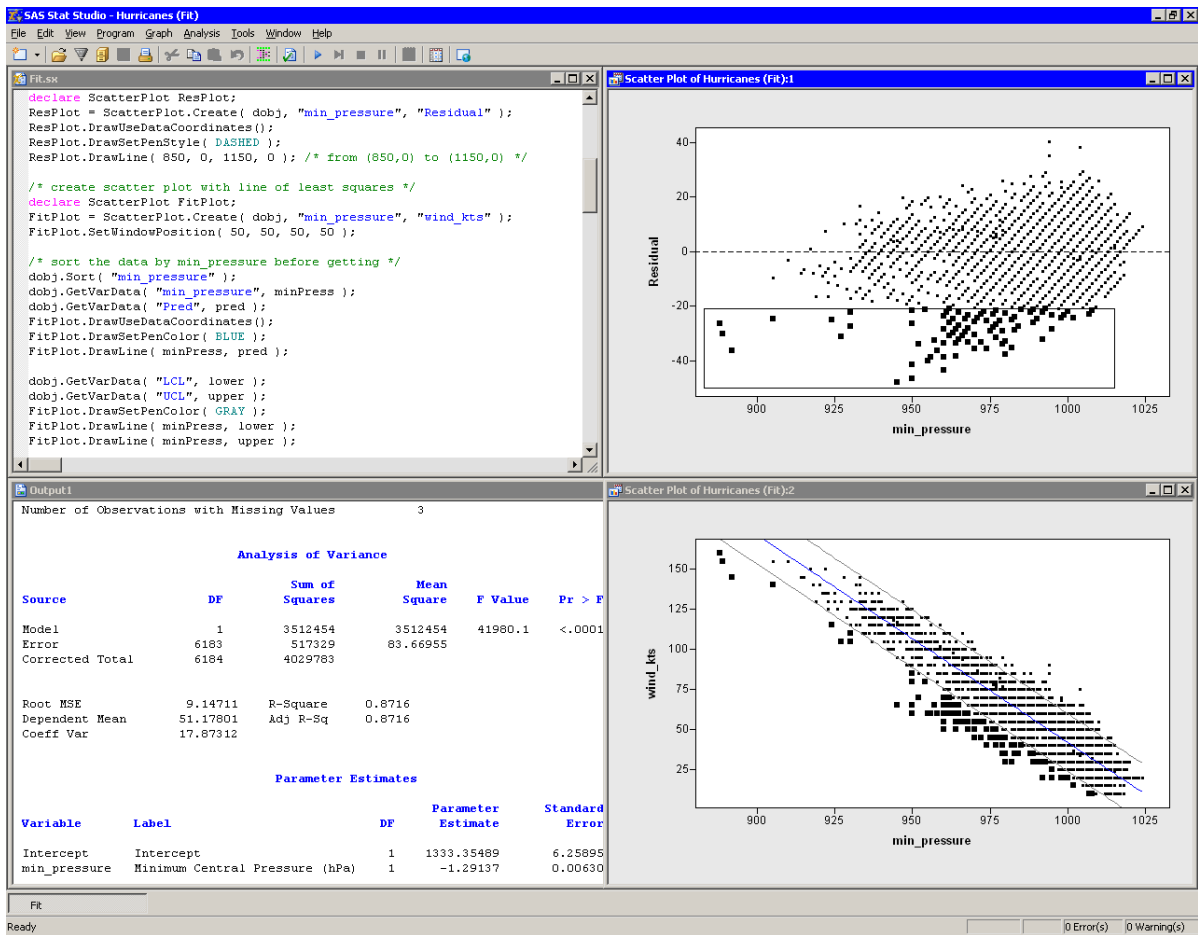
The predicted line is drawn in blue, as specified by the `DrawSetPenColor` method of the Plot class. The lower and upper limits of prediction are drawn in gray. All lines are drawn on top of the observations and are drawn as solid lines with a width of one pixel, since those default values were not changed. [Figure 6.2](#) shows the resulting plot.

Figure 6.2 A Fitted Model and Confidence Intervals



Notice that the preceding statements read variables from an output data set into a DataObject. This is possible only if the output data set has the same number of observations as the DataObject. The advantages of reading variables into the DataObject are that you can open a data table to see the values of the added variables and that you can use these variables in subsequent analyses. Furthermore, you can use the dynamically linked graphs to identify observations in a plot that have certain characteristics. For example, Figure 6.3 shows the observations that have large negative residuals for this model.

Figure 6.3 Observations with Large Negative Residuals



NOTE: Alternatively, you could have used the USE and READ statements to read the predicted values and the upper and lower limits of prediction into SAS/IML matrices. This approach is described in the next section.

Draw a Curve from Variables in a Data Set or from a SAS/IML Matrix

Sometimes the number of observations in your data is different from the number of observations in an output data set created by a procedure. This eliminates the possibility of reading the output into the DataObject that contains the original data.

For example, you can use the SCORE procedure to evaluate a linear model on a separate set of values of the explanatory variables. Or you can use the SCORE statement of the LOESS, GAM, or TPSPLINE procedure to evaluate a nonparametric model. In each of these cases, you can read variables from the output data set by using the SAS/IML USE and READ statements and then use the DrawLine method to add the relevant curve to a plot.

As an example of this approach, suppose you want to add a kernel density estimate to a histogram of the min_pressure variable. You can do this by calling the KDE procedure. Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
submit;
proc kde data=Hurr;
    univar min_pressure / out=KDEOut plots=none;
run;
endsubmit;
```

The KDE procedure creates the data set Work.KDEout on the SAS server with 401 observations. The data set contains a variable Value that consists of evenly spaced points between the minimum and maximum values of min_pressure. A variable named Density contains the kernel density estimate evaluated at the points of Value.

NOTE: You do not need to use the SORT procedure to sort this data set by Value, because it was created in sorted order. However, for unsorted data, you need to sort by the independent variable.

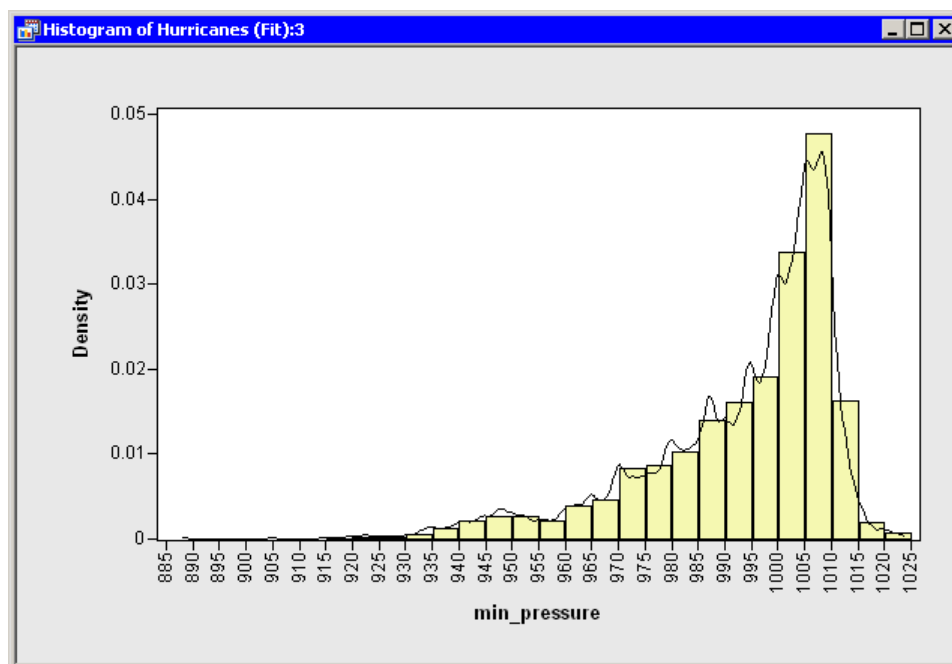
A straightforward approach is to create a histogram and add the kernel density estimate by reading in the Work.KDEout data set. Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
use KDEOut;
read all var {value density};
close KDEOut;

declare Histogram hist;
hist = Histogram.Create( dobj, "min_pressure" );
hist.ShowDensity(); /* show density instead of frequency */
hist.DrawUseDataCoordinates();
hist.DrawLine( value, density );
```

Figure 6.4 displays the resulting histogram and the overlaid kernel density estimate.

Figure 6.4 A Histogram Overlaid with a Kernel Density Estimate



Chapter 7

Reading ODS Tables

Contents

Introduction to Reading ODS Tables	43
Read Data from an ODS Table	43

Introduction to Reading ODS Tables

In Chapter 6, “Adding Curves to Graphs,” you generated output data sets by using SAS procedures and learned how to read variables from the server into the DataObject and into SAS/IML matrices. This chapter shows you how to read information from tables into an IMLPlus program.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ► Open ► File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the `ODS.sx` file.
- 5 Click **Open**.

Read Data from an ODS Table

The key to this chapter is knowing how to create SAS data sets directly from output tables by using the Output Delivery System (ODS). When you run a SAS procedure, the procedure typically creates one or more tables. By default, ODS sends the tables to the SAS listing, which in SAS/IML Studio is called the *output window*. But you can also tell ODS to send a table to a SAS data set. By subsequently reading the data set, you can read statistics from tables into your IMLPlus programs.

Suppose you perform a linear regression analysis with the REG procedure, and you want to read certain statistics into SAS/IML matrices. You might want to use the statistics in future computations, or you might want to use them in a title or legend for a plot.

To be specific, suppose you want to read the following statistics into your IMLPlus program: the number of observations used in the regression, the R-square value of the fitted model, and the slope and intercept of the least squares line.

Your first task is to determine the name of the ODS tables that contain the information you need. The “Details” section of the documentation for PROC REG includes a section about the names of ODS tables. You can also have ODS display the names by including the ODS TRACE ON statement before running your PROC REG statement.

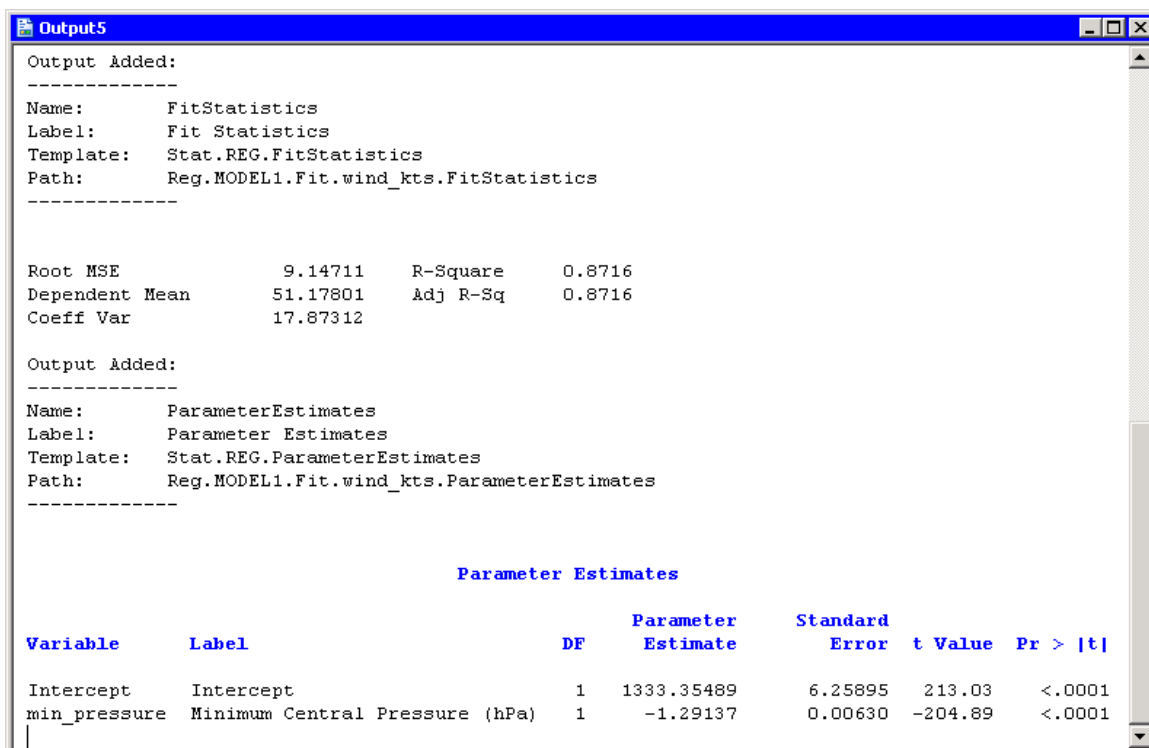
This approach is illustrated in the following program. Type or copy the following statements into the program window, and select **Program ► Run** from the main menu.

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");

dobj.WriteVarsToServerDataSet( {"wind_kts" "min_pressure"},
    "Work", "Hurr", true );

submit;
ods trace on / listing;
proc reg data=Hurr plots=none;
    model wind_kts = min_pressure;
run;
ods trace off;
endsubmit;
```

Figure 7.1 Printing ODS Tables Names



Output Added:

Name: FitStatistics
Label: Fit Statistics
Template: Stat.REG.FitStatistics
Path: Reg.MODEL1.Fit.wind_kts.FitStatistics

Root MSE	9.14711	R-Square	0.8716
Dependent Mean	51.17801	Adj R-Sq	0.8716
Coeff Var	17.87312		

Output Added:

Name: ParameterEstimates
Label: Parameter Estimates
Template: Stat.REG.ParameterEstimates
Path: Reg.MODEL1.Fit.wind_kts.ParameterEstimates

Parameter Estimates

Variable	Label	DF	Parameter Estimate	Standard Error	t Value	Pr > t
Intercept	Intercept	1	1333.35489	6.25895	213.03	<.0001
min_pressure	Minimum Central Pressure (hPa)	1	-1.29137	0.00630	-204.89	<.0001

The output is shown in [Figure 7.1](#). The LISTING option in the ODS TRACE ON statement tells ODS to send the names of tables to the LISTING destination rather than to the SAS log. By scrolling through the output, you can see that the number of observations used in the regression is part of the NObs table. The R-square value of the fitted model is contained in the FitStatistics table. The slope and intercept of the least squares line are contained in the ParameterEstimates table. The following example uses the ODS OUTPUT statement to write these tables to data sets in Work.

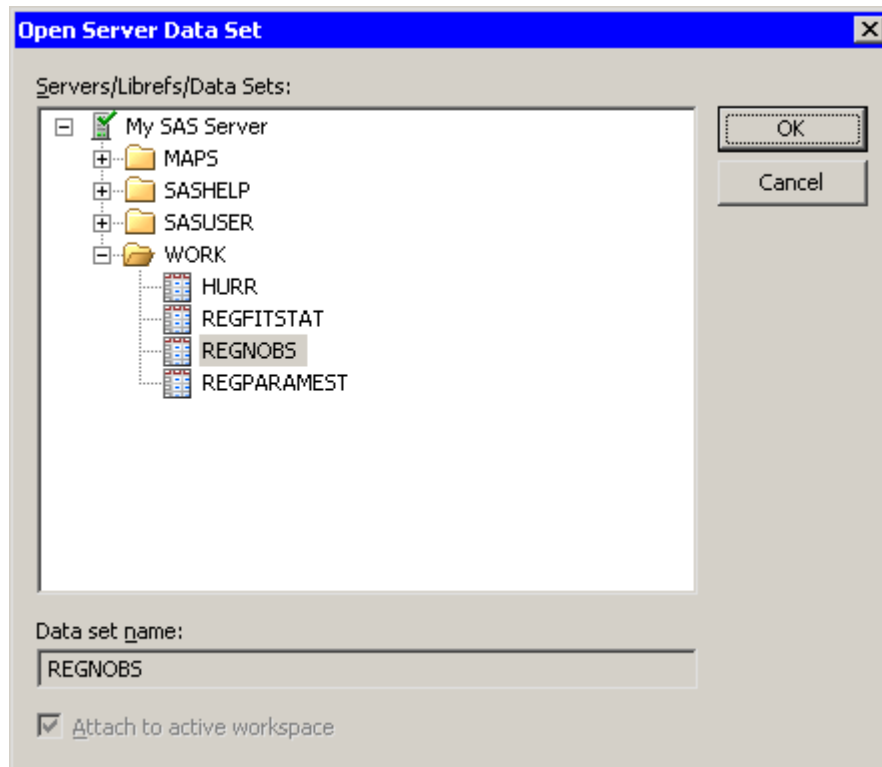
Replace the SUBMIT/ENDSUBMIT block of statements in the program window with the following revised statements. Then select **Program ► Run** from the main menu.

```
submit;
proc reg data=Hurr plots=none;
  model wind_kts = min_pressure;
  ods output NObs = RegNObs
             FitStatistics = RegFitStat
             ParameterEstimates = RegParamEst;
run;
endsubmit;
```

These statements cause the REG procedure to create three new data sets in the Work library. You can examine each data set by opening it from the server into a data table as follows:

- 1 Select **File ► Open ► Server Data Set** from the main menu. The dialog box in [Figure 7.2](#) appears.
- 2 Expand the entry for your current SAS server.
- 3 Expand the entry for the Work directory.
- 4 Click the RegNObs data set.
- 5 Click **OK**.
- 6 Repeat the previous steps to open the RegFitStat and RegParamEst data sets.

Figure 7.2 Opening a Server Data Set



You might notice that some data sets look different from the corresponding tables that are displayed in the output window. Some tables have nonprinting columns that are not visible in the output window but *are* written to the data set.

By opening each data set in turn, you can determine the name of the variable that contains the information that you want to read into SAS/IML matrices. The number of observations used in the regression is contained in the `N` variable in the `RegNObs` data set. The R-square value of the fitted model is contained in the `NValue2` variable in the `RegFitStat` data set. The slope and intercept of the least squares line are contained in the `Estimate` variable of the `RegParamEst` data set.

The SAS/IML language provides statements to read and write server data to and from matrices. The `USE` statement opens a SAS data set, and the `READ` statement reads server data into matrices. Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```

use RegNObs;
read all var {N};
close RegNObs;

use RegFitStat;
read all var {Label2 NValue2}; /* name of statistic and value */
close RegFitStat;

use RegParamEst;
read all var {Variable Estimate}; /* var name and coefficient */
close RegParamEst;

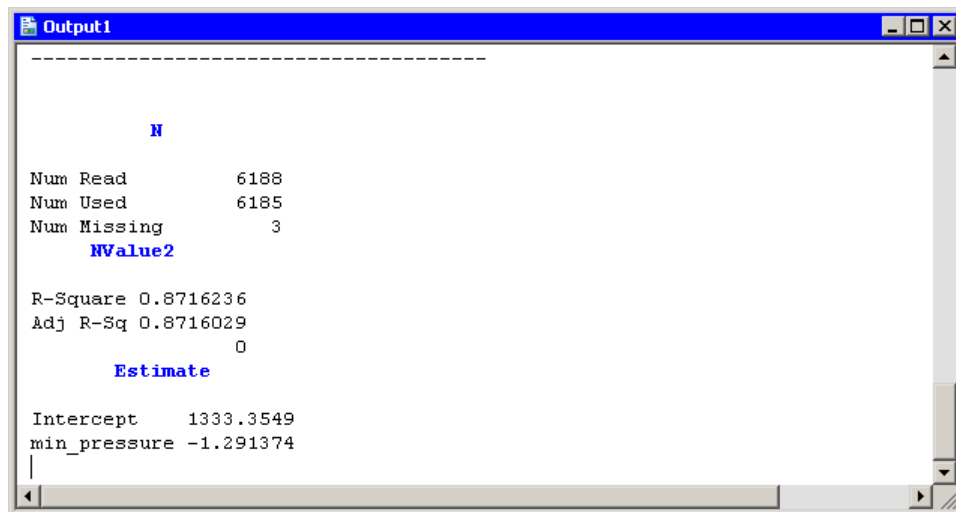
print N[ rowname={"Num Read", "Num Used", "Num Missing"} ],
      NValue2[ rowname=Label2 ],
      Estimate[ rowname=Variable ];

```

The output is shown in [Figure 7.3](#). Statistics from the REG tables are now contained in SAS/IML matrices. You can use these matrices in computations or, as the next chapter shows, in a title or legend for a plot.

NOTE: You can often read in the names of statistics from the ODS data set, as the **Label2** and **Variable** matrices in the preceding example demonstrate.

Figure 7.3 Creating Matrices from ODS Tables



Chapter 8

Adding Titles, Legends, and Insets

Contents

Introduction to Adding Titles, Legends, and Insets	49
Create Statistics for the Title and Legends	49
Create a Title Based on an Analysis	51
Create a Legend	52
Create an Inset Based on an Analysis	53

Introduction to Adding Titles, Legends, and Insets

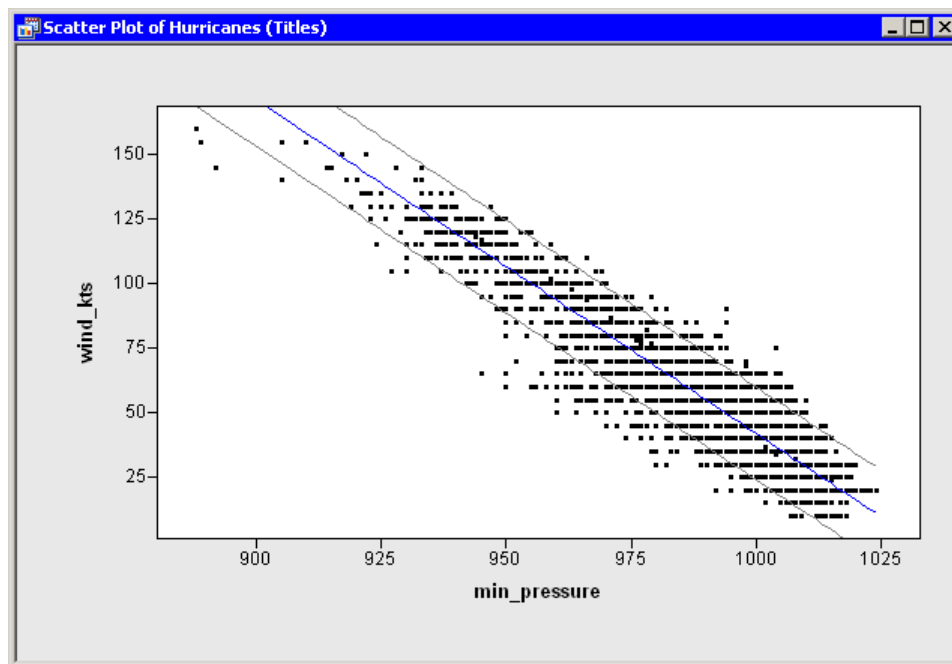
In Chapter 7, “[Reading ODS Tables](#),” you learned how to write ODS tables to SAS data sets and then how to read variables from the data sets into SAS/IML matrices. This chapter shows you how to create titles, legends, and insets on a graph. (An *inset* is a box that contains statistics and is drawn on the plot.) In particular, you learn how to incorporate statistical information from ODS tables into titles, legends, and insets.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ►Open ►File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the `Titles.sx` file.
- 5 Click **Open**.

Create Statistics for the Title and Legends

In previous chapters you performed a linear regression with PROC REG, and you created a scatter plot that shows the fitted model and 95% confidence intervals for individual predictions as shown in [Figure 8.1](#).

Figure 8.1 A Fitted Model and Confidence Intervals

Suppose now that you want to add a title and legends to the graph that results from the linear regression. Specifically, you want to display the following information:

- a title that shows the fitted model
- a legend that explains the different colors of lines in the plot
- an inset that shows the number of observations used in the regression and the R-square value of the fitted model

In Chapter 7, “[Reading ODS Tables](#),” you learned how to get all of the needed information out of the ODS tables output by using PROC REG. The program that follows combines ideas from previous chapters. Copy the following statements into a program window, and select **Program ► Run** from the main menu.

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");

dobj.WriteVarsToServerDataSet( {"wind_kts" "min_pressure"},
    "Work", "Hurr", true );

submit;
proc reg data=hurr plots=none;
    model wind_kts = min_pressure;
    output out=RegOut P=Pred LCL=LCL UCL=UCL;
    ods output NObs = RegNObs
        FitStatistics = RegFitStat
        ParameterEstimates = RegParamEst;
run;
endsubmit;
```

```

ok = CopyServerDataToDataObject( "Work", "RegOut", dobj,
    {"Pred" "LCL" "UCL"}, /* names on server */
    {"Pred" "LCL" "UCL"}, /* names in DataObject */
    {"Predicted" "Lower Conf. Limit" "Upper Conf. Limit"},
    true );

declare ScatterPlot FitPlot;
FitPlot = ScatterPlot.Create( dobj, "min_pressure", "wind_kts" );

dobj.Sort( "min_pressure" );
dobj.GetVarData( "min_pressure", minPress );
dobj.GetVarData( "Pred", pred );
FitPlot.DrawUseDataCoordinates();
FitPlot.DrawSetPenColor( BLUE );
FitPlot.DrawLine( minPress, pred );

dobj.GetVarData( "LCL", lower );
dobj.GetVarData( "UCL", upper );
FitPlot.DrawSetPenColor( GRAY );
FitPlot.DrawLine( minPress, lower );
FitPlot.DrawLine( minPress, upper );

use RegNObs;      read all var {N};      close RegNObs;
use RegFitStat;  read all var {NValue2}; close RegFitStat;
use RegParamEst; read all var {Estimate}; close RegParamEst;

print N[ rowname={"Num Read", "Num Used", "Num Missing"} ],
      NValue2[ rowname={"R-Square", "Adj R-sq", " " } ],
      Estimate[ rowname={"Intercept", "min_pressure"} ];

```

Create a Title Based on an Analysis

This section adds a title to the graph that shows the fitted model. You can add a title by calling the `SetTitleText` and `ShowTitle` methods of the `Plot` class. However, in this case you first need to convert the values stored in the **Estimate** matrix from numerical to character values. One way to do this is to apply a *w.d* format by using the Base SAS function `PUTN`. You can then concatenate pieces of the title together using the SAS/IML `CONCAT` function and display the result.

The statements that follow build the title intelligently. Let b_0 be the intercept and b_1 be the slope of the regression line. The regression line for this example has negative slope ($b_1 < 0$). If you built the title as

```
title = concat("wind_kts = ", b0, " + ", b1, " * min_pressure");
```

then the title string might appear as

```
wind_kts = 1333.35 + -1.29 * min_pressure
```

While this is correct, it is awkward to read. A more aesthetic title would be

```
wind_kts = 1333.35 - 1.29 * min_pressure
```

This can be accomplished by treating the case of negative slope separately from the case of positive slope.

Add the following statements at the bottom of the program window, and select **Program ►Run** from the main menu. Figure 8.2 shows how the title appears.

```
b0 = putn( Estimate[1], "7.2" );
if Estimate[2]<0 then do;          /* if slope is negative */
    sign = " - ";                  /* display b0 - (-b1) */
    b1 = putn( -Estimate[2], "4.2" );
end;
else do;                          /* else display b0 + b1 */
    sign = " + ";
    b1 = putn( Estimate[2], "4.2" );
end;
title = concat( "wind_kts = ", b0, sign, b1, " * min_pressure" );
FitPlot.SetTitleText( title );
FitPlot.ShowTitle();
```

Create a Legend

This section adds a legend that explains the different colors of lines in the graph. Adding a legend is usually accomplished by using the DrawLegend module that is distributed with SAS/IML Studio. The module is documented in the SAS/IML Studio online Help. The statements that follow show one choice for displaying a legend. Add these at the bottom of the program window, and select **Program ►Run** from the main menu. Figure 8.2 shows how the legend appears.

```
Labels = {"Least-Squares Fit" "95% Prediction Limits"};
LabelSize = 8;          /* 8 point font */
LineColor = BLUE || GRAY; /* form 1x2 matrix */
LineStyle = SOLID;      /* all lines are solid */
Symbol = -1;            /* no symbols */
BackgroundColor = WHITE;
Location = 'ILB';        /* Inside, Left, Bottom */
run DrawLegend( FitPlot, Labels, LabelSize,
                LineColor, LineStyle, Symbol,
                BackgroundColor, Location );
```

Note the use of the SAS/IML concatenation operator `||`. This operator takes the two predefined integers BLUE and GRAY and concatenates them into a 1×2 matrix.

NOTE: Trying to form the matrix by using the following statement does not work because the SAS/IML language interprets that statement as forming a character matrix with values “BLUE” and “GRAY”:

```
LineColor = {BLUE GRAY}; /* wrong! */
```

Similarly, as the next statement illustrates, you must use the concatenation operator when creating a matrix from submatrices. You cannot write

```
Values = {N[2] NValue2[1]}; /* wrong! */
```

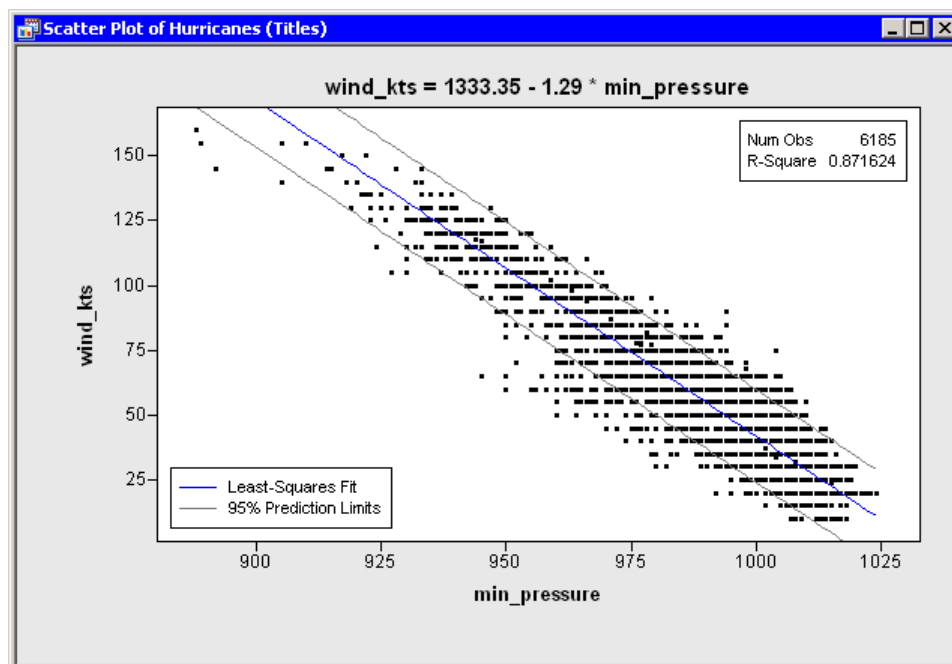
Create an Inset Based on an Analysis

This section adds an inset to the graph that shows the number of observations used in the regression and the R-square value of the fitted model. You can add an inset by using the DrawInset module that is distributed with SAS/IML Studio. The module is documented in the SAS/IML Studio online Help. The following statements show one choice for creating an inset. Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
Labels = {"Num Obs" "R-Square"};
Values = N[2] || NValue2[1];
LabelProps = .; /* accept default settings for labels */
LabelTypeface = "Arial"; /* font */
BackgroundColor = -1; /* no color (transparent) */
Location = 'IRT'; /* Inside, Right, Top */
run DrawInset( FitPlot, Labels, Values,
              LabelProps, LabelTypeface,
              BackgroundColor, Location );
```

Figure 8.2 shows the title, the legend, and the inset created in this chapter.

Figure 8.2 Plot with Title, Legend, and Inset



Chapter 9

Adjusting Axes and Locations of Ticks

Contents

Introduction to Adjusting Axes and Locations of Ticks	55
Change Axes Properties and Tick Locations	55
Set Nonuniform Tick Locations	58
Adjust Tick Marks for a Histogram	60

Introduction to Adjusting Axes and Locations of Ticks

In Chapter 3, “Creating Dynamically Linked Graphs,” you created standard statistical graphs. In this chapter, you learn how to adjust the range and location of tick marks on axes. You also learn how to change the label for an axis. You cannot adjust the tick locations for nominal variables (for example, the horizontal axis on a bar chart), but you can change the axis label.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ►Open ►File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the `Axes.sx` file.
- 5 Click **Open**.

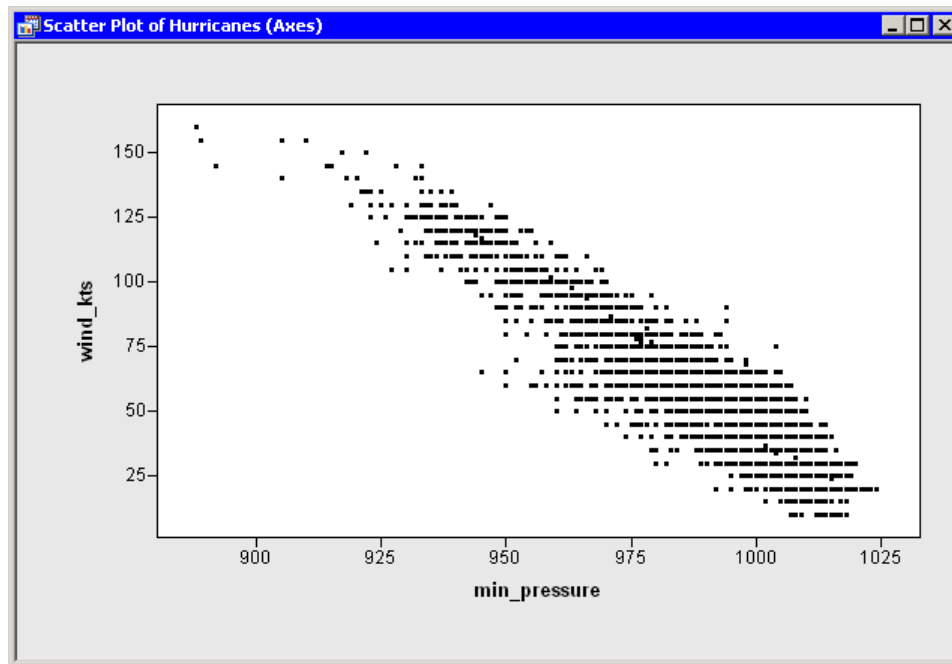
Change Axes Properties and Tick Locations

First, open the Hurricanes data set and create a scatter plot. Type or copy the following statements into the program window, and select **Program ►Run** from the main menu.

```
declare DataObject dobj;  
dobj = DataObject.CreateFromFile("Hurricanes");  
  
declare ScatterPlot plot;  
plot = ScatterPlot.Create( dobj, "min_pressure", "wind_kts" );
```

The plot in Figure 9.1 appears. SAS/IML Studio tries to make appropriate choices for the location of ticks and for the minimum and maximum of the axis view range. However, sometimes you might want to change characteristics of the axes.

Figure 9.1 Default Axes



There are several methods in the Plot class that can be used to change the way that axes are displayed. These methods all begin with the prefix `SetAxis`. For example:

- The `SetAxisLabel` method changes the axis label.
- The `SetAxisTickUnit` method changes the increment between tick marks.
- The `SetAxisTickAnchor` method shifts the location of tick marks along an axis. (The argument to `SetTickAnchor` must be a number between the minimum and maximum values of the axis tick range.)
- The `SetAxisMinorTicks` method adds minor tick marks (that is, unlabeled tick marks between major ticks).

These and other methods that affect axes are documented in the SAS/IML Studio online Help.

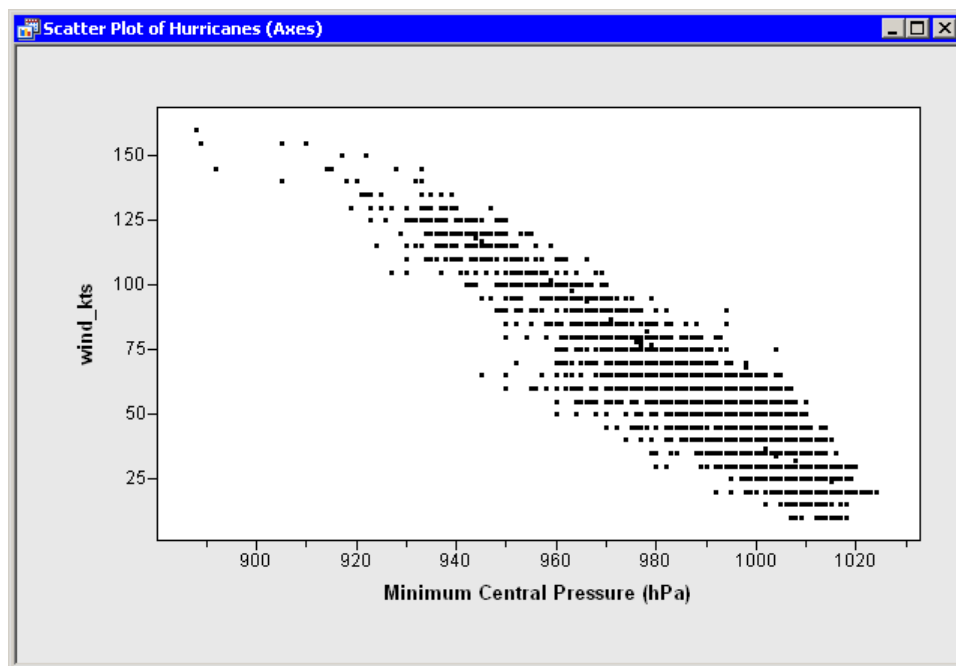
Suppose you want to change the horizontal axis in the following ways:

- Label the axis with the variable's label instead of the variable's name.
- Change the size of the increment between successive ticks from 25 to 20 so that the ticks marks are located at 900, 920, ..., 1020.
- Add one minor tick mark between each major tick.

To make these changes, add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu. Figure 9.2 shows the new label, minor ticks, and tick placement for the horizontal axis.

```
plot.SetAxisLabel ( XAXIS, AXISLABEL_VARLABEL );
plot.SetAxisTickUnit( XAXIS, 20 );
plot.SetAxisTickAnchor( XAXIS, 900 );
plot.SetAxisMinorTicks( XAXIS, 1 );
```

Figure 9.2 New Horizontal Axis

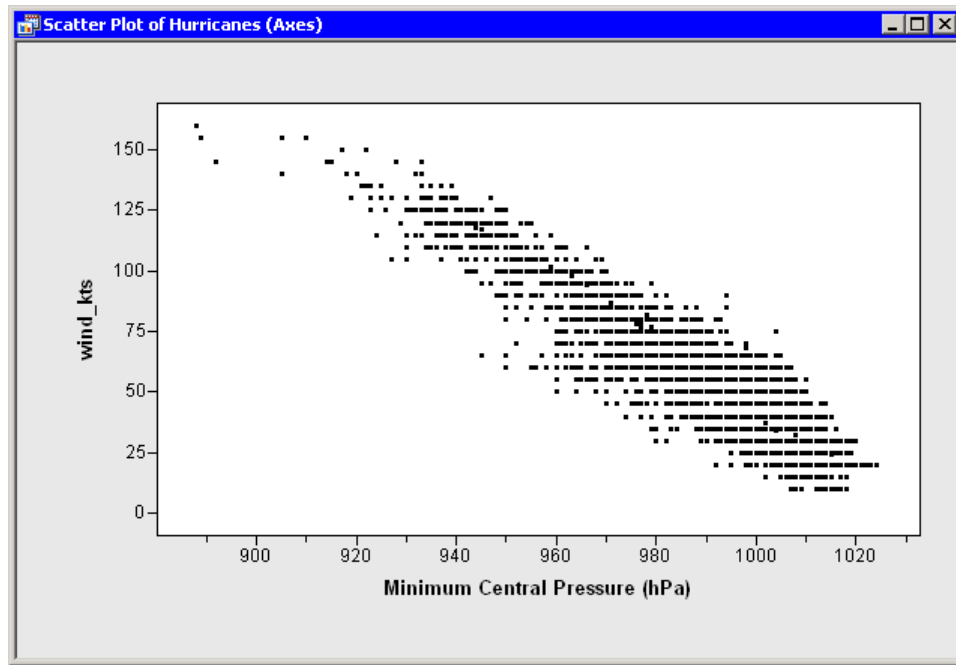


By default, the *axis view range* is determined by the minimum and maximum values of the axis variable. (More precisely, it is determined by the minimum and maximum values that are included in the plots.) At times you might want to override that default. A common reason for doing this is to include a reference value (for example, zero) in the plot, even though there are no observations with that value.

As an example of changing the axis view range, suppose you want the vertical axis (the `wind_kts` axis) to show zero wind speed as in [Figure 9.3](#). To change the ticks on the horizontal axis, add the following statement at the bottom of the program window, and select **Program ► Run** from the main menu.

```
plot.SetAxisViewRange( YAXIS, 0, 160 );
```

Figure 9.3 Vertical Axis Range That Includes Zero



Set Nonuniform Tick Locations

Up to this point, this example has shown you how to construct uniformly spaced tick marks at locations of your choosing. You can also draw tick marks that are not uniformly spaced by using the `SetAxisTicks` method of the `Plot` class. This method can also be used to display labels for ticks that are different from the numerical values of the ticks. For example, you might decide that you want the tick marks on the vertical axis to show the categories of the Saffir-Simpson intensity scale ([Table 9.1](#)).

Table 9.1 The Saffir-Simpson Intensity Scale

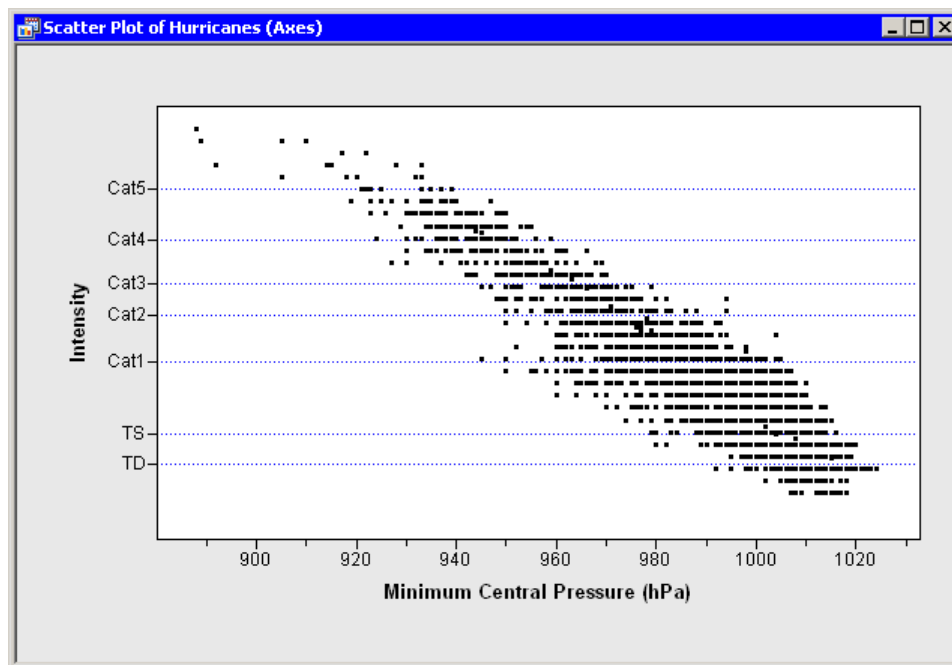
Category	Wind Speed (knots)
Tropical depression (TD)	22–34
Tropical storm (TS)	34–64
Category 1 hurricane (Cat1)	64–83
Category 2 hurricane (Cat2)	83–96
Category 3 hurricane (Cat3)	96–114
Category 4 hurricane (Cat4)	114–135
Category 5 hurricane (Cat5)	greater than 135

Add the following statements at the bottom of the program window, and select **Program ►Run** from the main menu. Figure 9.4 shows the tick locations and labels for the vertical axis.

```

plot.SetAxisLabel( YAXIS, "Intensity" );
StormTicks = {22 34 64 83 96 114 135};
StormLabels = {'TD' 'TS' 'Cat1' 'Cat2' 'Cat3' 'Cat4' 'Cat5'};
plot.SetAxisTicks( YAXIS, StormTicks, StormLabels );
plot.ShowAxisReferenceLines( YAXIS );

```

Figure 9.4 Vertical Axis with Custom Ticks

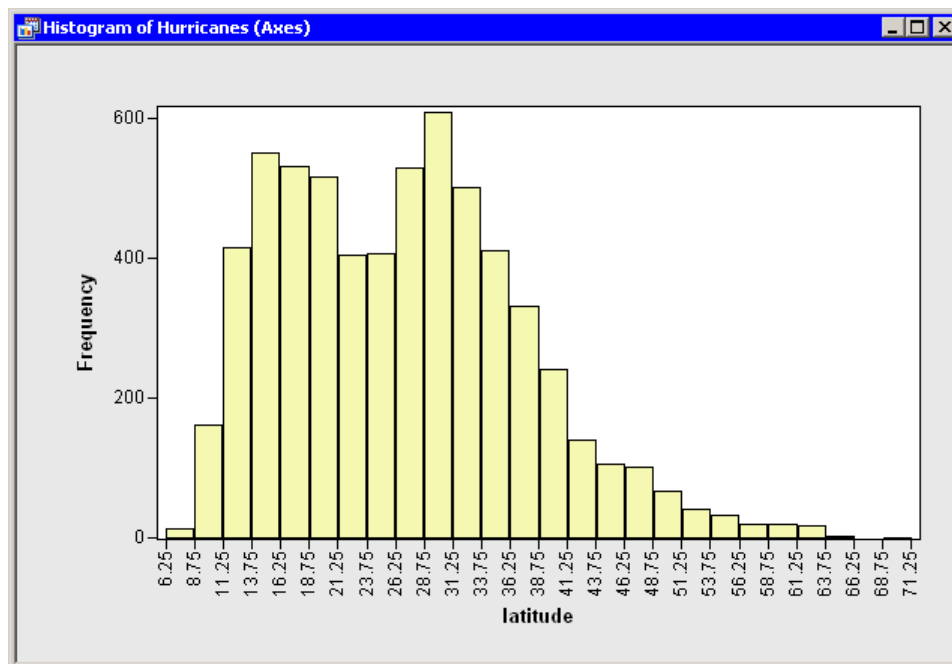
Adjust Tick Marks for a Histogram

For a histogram, the location of tick marks determines the bins used to visualize the frequency distribution. Therefore, the Histogram class has some special rules for specifying tick marks.

Create a histogram by adding the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
declare Histogram hist;
hist = Histogram.Create( dobj, "latitude" );
hist.SetWindowPosition( 50, 50, 50, 50 );
```

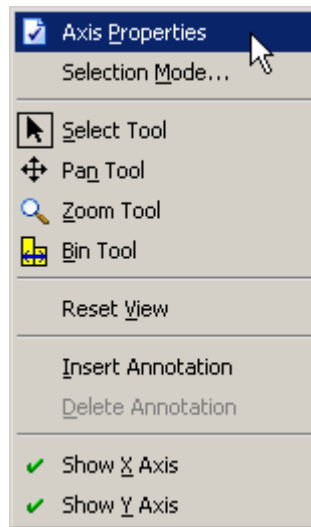
Figure 9.5 Histogram with Default Bins



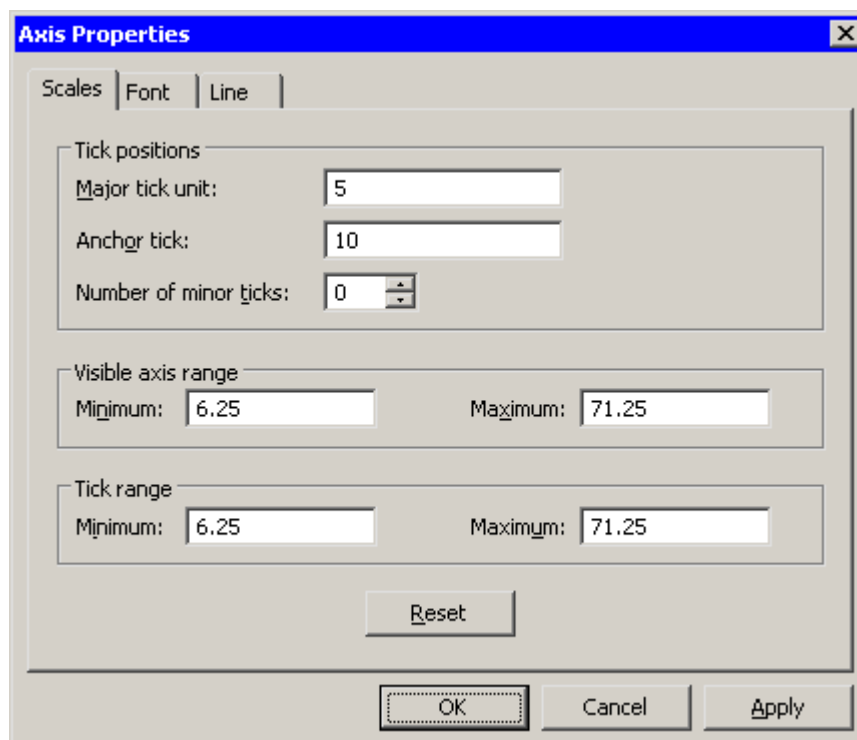
These statements create a histogram from the `latitude` variable as shown in [Figure 9.5](#). To avoid having the histogram appear on top of the scatter plot, the `SetWindowPosition` method is called to move the histogram into the lower right corner of the SAS/IML Studio workspace.

For a histogram, the major tick unit is also the width of each histogram bin. The tick marks for this histogram are anchored at 6.25 and have a tick unit of 2.5. The following steps show you how to change the location of the histogram ticks so that the bins show the frequency of observations in the intervals 5–10, 10–15, 15–20, and so on.

- 1 Right-click on the horizontal axis of the histogram. A pop-up menu appears as in [Figure 9.6](#).

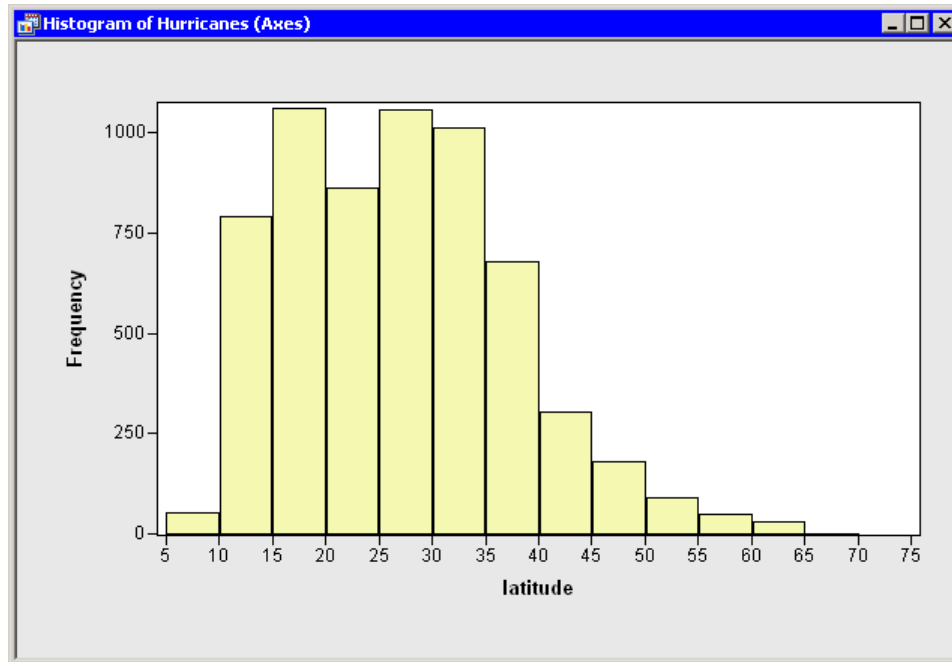
Figure 9.6 Plot Area Pop-up Menu

- 2 Select **Axis Properties** from the pop-up menu. The Axis Properties dialog box appears as in [Figure 9.7](#). This is a quick way to determine the anchor location, tick unit, and tick range for an axis.
- 3 Change the value in the **Major tick unit** field to 5.
- 4 Change the value in the **Anchor tick** field to 10.
- 5 Click **OK**.

Figure 9.7 Specifying Histogram Bins

The histogram updates to reflect the new histogram bin locations (Figure 9.8). The visible axis range and the tick range (both shown in Figure 9.7) are automatically widened, if necessary, so that all histogram bins are visible.

Figure 9.8 Histogram with Customized Bins



You can also change the location and width of histogram bins by using program statements. For example, the Histogram class has a special `ReBin` method that you can often use to change the anchor tick and the major tick unit in a single method call. However, in the current example, you do not get tick marks at 0 or at 75 if you use only the following statement:

```
hist.ReBin( 10, 5 );
```

This is because no tick mark can be outside the interval specified by the **Tick Range** values in Figure 9.7, and the tick range for this example is [6.25, 71.25]. You can adjust the axis tick range by using the `SetAxisTickRange` method of the Plot class:

```
hist.SetAxisTickRange( XAXIS, 0, 75 );
hist.ReBin( 10, 5 );
```

Another solution is to use the `SetAxisNumericTicks` method of the Plot class to change the bin locations of a histogram. This method requires that you specify the tick anchor, unit, and range, all in the same call.

```
hist.SetAxisNumericTicks( XAXIS, 10, 5, 0, 75 );
```

Chapter 10

Changing the Color and Shape of Observation Markers

Contents	
Introduction to Changing the Color and Shape of Markers	63
Color Markers to Represent Values of a Continuous Variable	63
Color Markers to Represent Values of a Nominal Variable	65
Color Markers That Satisfy a Criterion	66
Change Marker Shapes	68
Show Only Selected Observations	69

Introduction to Changing the Color and Shape of Markers

In Chapter 3, “Creating Dynamically Linked Graphs,” you created standard statistical graphs. In this chapter you learn how to change the appearance of observation markers. You can change the color and shape of markers. For each plot, you can also choose the size of markers, and whether a marker is displayed all the time or only when it is selected.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ► Open ► File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the *Markers.sx* file.
- 5 Click **Open**.

Color Markers to Represent Values of a Continuous Variable

You might find it useful to color observation markers by using the value of an interval (that is, continuous) variable. This enables you to examine values of the coloration variable, even if that variable is not being explicitly plotted.

Suppose you are looking at a scatter plot of the `min_pressure` and `wind_kts` variables in the Hurricanes data set. You want to color-code observations by using the value of a third variable, latitude. You want to assign red to the most southerly observation, blue to the most northerly, and other colors to observations with intermediate values.

To accomplish this, you can call the `ColorCodeObs` module, which is distributed with SAS/IML Studio. The module colors observations according to values of a single variable by using a user-defined color blend. Type or copy the following statements into the program window, and select **Program ► Run** from the main menu.

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");

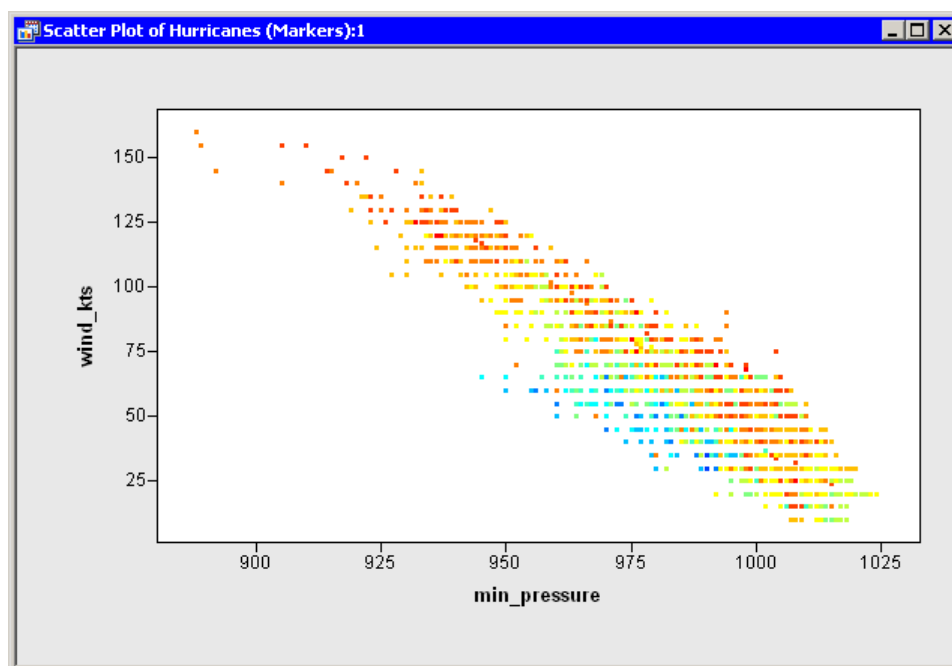
declare ScatterPlot plot;
plot = ScatterPlot.Create( dobj, "min_pressure", "wind_kts" );

ColorMap = RED//YELLOW//CYAN//BLUE;
NumColors = 13;
run ColorCodeObs( dobj, "latitude", ColorMap, NumColors );

declare Histogram hist;
hist = Histogram.Create( dobj, "latitude" );
hist.SetWindowPosition( 50, 50, 50, 50 );
hist.SetAxisNumericTicks( XAXIS, 10, 5, 0, 75 );
```

The first and last statements create a scatter plot and histogram as described in Chapter 3, “Creating Dynamically Linked Graphs.” They also adjust the histogram bins as described in Chapter 9, “Adjusting Axes and Locations of Ticks.” The resulting scatter plot is shown in Figure 10.1.

Figure 10.1 Markers Colored by an Interval Variable



The new statements in this program are those that color markers by using the `latitude` variable. This is done with the `ColorCodeObs` module. In this example, the `latitude` variable is used to color observations. The smallest value of the `latitude` variable (7.2) is assigned to the first color (red) in the `ColorMap` matrix. The largest value of the `latitude` variable (70.7) is assigned to the last color (blue) in the `ColorMap` matrix. The remaining values are assigned to one of 13 colors obtained by linearly blending the four colors defined in the `ColorMap` matrix. Observations are colored yellow if they are near 28.4 degrees ($28.4 \approx 7.2 + \frac{1}{3}(70.7 - 7.2)$). Observations are colored cyan if they are near 49.5 degrees ($49.5 \approx 7.2 + \frac{2}{3}(70.7 - 7.2)$).

NOTE: Observations with missing values for the requested variable are not colored. The `latitude` variable used in this example does not contain any missing values.

To confirm that the observations were color-coded according to values of `latitude`, follow these steps:

- 1 Click on the histogram bars for low values of `latitude`.

Note that the observations are mainly colored red and orange. Orange appears because the red and yellow colors in the `ColorMap` matrix were blended.

- 2 Click on the histogram bars for high values of `latitude`.

High values of `latitude` are colored in shades of blue. Each shade is a blend of cyan and blue.

- 3 Click on the histogram bars for medium values of `latitude`.

Medium values of `latitude` are colored in shades of yellows and greens.

You can use the predefined colors available in SAS/IML Studio, or you can create your own colors by specifying their RGB or hexadecimal values as described in the SAS/IML Studio online Help. [Table 10.1](#) lists the predefined colors in SAS/IML Studio. Each color (written in all capitals) is an IMLPlus keyword.

Table 10.1 Predefined Colors

BLACK	CHARCOAL	GRAY, GREY	WHITE
RED	MAROON	SALMON	ROSE
MAGENTA	PURPLE	LILAC	PINK
BLUE	STEEL	VIOLET	CYAN
GREEN	OLIVE	LIME	
YELLOW	ORANGE	GOLD	
BROWN	TAN	CREAM	

Color Markers to Represent Values of a Nominal Variable

You might also find it useful to color observation markers according to the value of a nominal (that is, discrete) variable. This enables you to visually identify specific categories of the coloration variable, provided that the variable has a small number of categories.

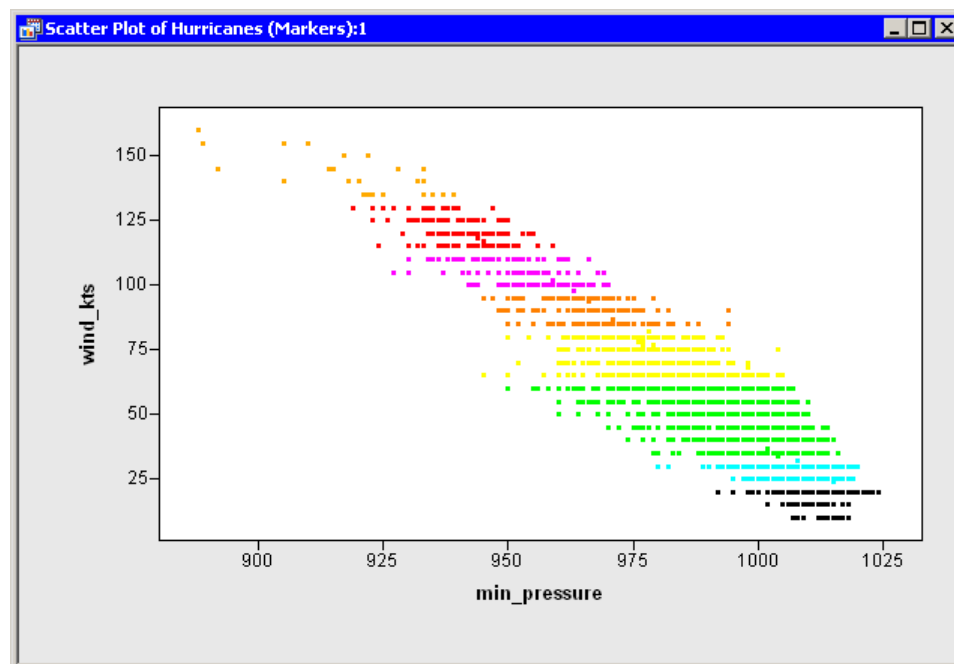
You can use the `ColorCodeObsByGroups` module to color observation markers according to the value of a nominal variable. One of the arguments to the `ColorCodeObsByGroups` module is a vector of colors (a *color map* or *color ramp*) that has the same number of colors as the number of unique categories in the nominal coloration variable.

If the coloration variable is a character variable, then color coding corresponds to an alphabetical ordering of the values of the variable. The first color corresponds to the first sorted value; the last color corresponds to the last sorted value. If the coloration variable is a numeric nominal variable, then color coding corresponds to a numeric ordering of the values of the variable. Missing values appear first in the sorted order for all variables.

For example, in the Hurricanes data set, you might want to color observations by using the category variable that contains the Saffir-Simpson intensity of the cyclone as given in Table 9.1. Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu. The resulting plot is shown in Figure 10.2.

```
/* color markers by value of a nominal variable */
/*      missing Cat1   Cat2   Cat3   Cat4 Cat5   TD   TS */
ColorMap = BLACK//YELLOW//ORANGE//MAGENTA//RED//GOLD//CYAN//GREEN;
run ColorCodeObsByGroups( dobj, "category", ColorMap );
```

Figure 10.2 Markers Colored by a Nominal Variable



Color Markers That Satisfy a Criterion

The ColorCodeObs and ColorCodeObsByGroups modules color observations by calling the SetMarkerColor method of the DataObject class. While these module are often convenient to use, sometimes you might want to color markers according to some criterion that is not covered by either module. In this case, you can use the SetMarkerColor method directly.

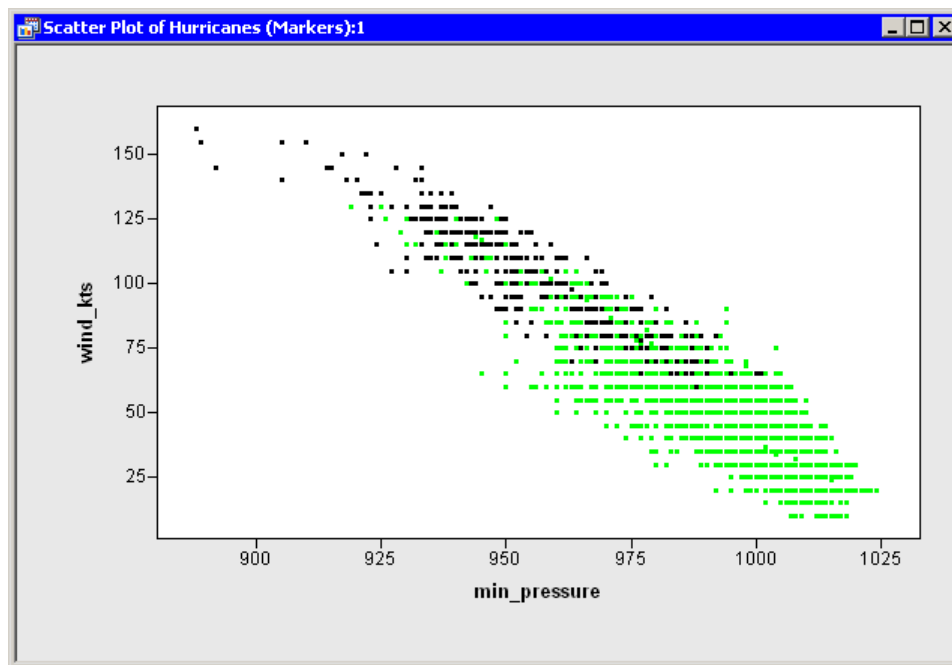
The SetMarkerColor method changes the colors of specified observations. It is usually used in conjunction with the GetVarData method of the DataObject and the SAS/IML LOC function to find observations that satisfy some criterion.

As an example, suppose you are interested in the size of the eye of a tropical cyclone. (The eye of a cyclone is a calm, relatively cloudless central region.) The Hurricanes data set has a variable `radius_eye` that gives the radius of the cyclone's eye in nautical miles. The `radius_eye` variable has many missing values, because not all storms have well-defined eyes. You can use the `SetMarkerColor` method to visualize the observations for which well-defined eyes exist.

Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu.

```
/* color markers manually */
doobj.SetMarkerColor( OBS_ALL, BLACK );
doobj.GetVarData( "radius_eye", rEye );
idx = loc( rEye = . );
if ncol(idx)>0 then
    doobj.SetMarkerColor( idx, GREEN );
```

Figure 10.3 Markers Colored Manually



The first statement uses the special keyword `OBS_ALL` to set the color of all observations in the data set to black. The `GetVarData` method of the `DataObject` copies the values of the `radius_eye` variable into a SAS/IML matrix called `rEye`. The `LOC` function is then used to find the indices of missing values in the `rEye` matrix. If at least one element of the matrix satisfies the condition, then the `SetMarkerColor` method sets the color of corresponding observations to green.

You can see from the resulting graph (Figure 10.3) that very few storms with low wind speeds have well-defined eyes, whereas most of the intense storms have well-defined eyes.

Of course, you could repeat this process for other conditions you want to visualize. For example, you could locate the values of `rEye` that are greater than or equal to 20 nautical miles and color those observations.

Change Marker Shapes

When a graph is printed on a gray-scale printer, it is often easier to discern observations that have different marker shapes than it is to discern markers of different colors. Even on a computer screen, marker shape is sometimes preferred for classifying markers according to a small number of discrete values. For example, if some observations represent males and others females, marker shape is an ideal way to encode that information.

Just as you can change marker colors with the `SetMarkerColor` method, you can change a marker's shape with the `DataObject`'s `SetMarkerShape` method. For the Hurricanes data set, suppose you want to use marker shape to differentiate observations with missing values for the `radius_eye` variable from those with nonmissing values. One way to accomplish this is to copy the values of `radius_eye` from the `DataObject`, and then use the `LOC` function to find the observation numbers with certain properties. You can then use the `SetMarkerShape` method to set the shape of the observations.

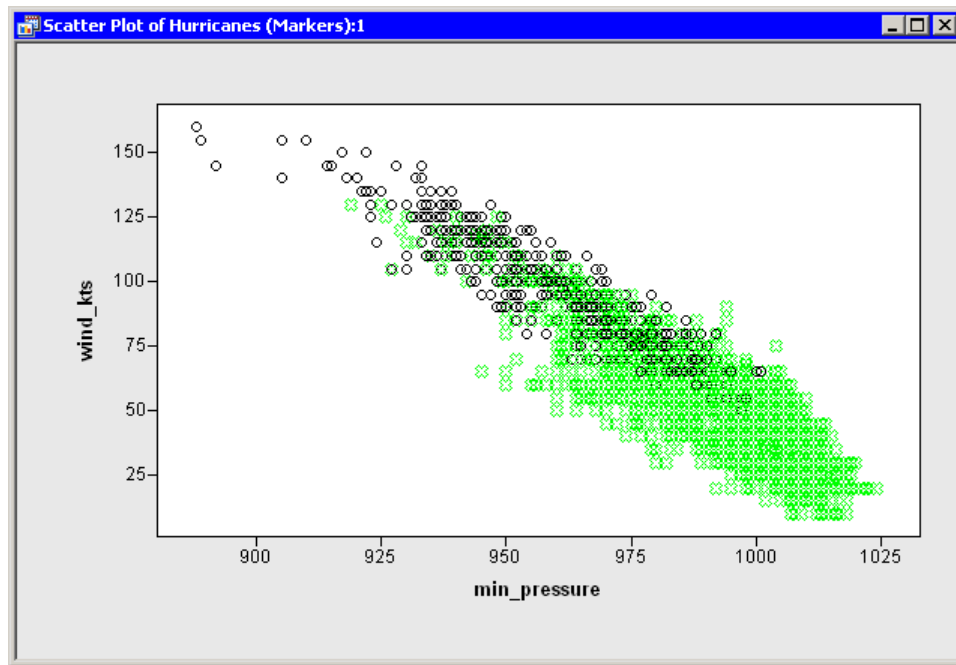
Add the following statements at the bottom of the program window, and select **Program ► Run** from the main menu. Figure 10.4 shows the result of running these statements.

```
/* change marker shapes */
dobj.GetVarData("radius_eye", rEye );
idx = loc( rEye = . );
if ncol(idx)>0 then
    dobj.SetMarkerShape( idx, MARKER_X );
idx = loc( rEye ^= . );
if ncol(idx)>0 then
    dobj.SetMarkerShape( idx, MARKER_CIRCLE );
plot.SetMarkerSize( 6 );
dobj.SetMarkerFillColor( OBS_ALL, NOCOLOR );
```

When you run these statements, the `GetVarData` method of the `DataObject` copies the values of the `radius_eye` variable into a SAS/IML matrix called `rEye`. The `LOC` function is then used to find the indices of missing values in the `rEye` matrix. If at least one element of the matrix satisfies the condition, then the `SetMarkerShape` method sets the shape of corresponding observations to an “x.” The shape of markers for observations with nonmissing values is set to a circle.

NOTE: It is a good programming practice to verify that the `LOC` statement did not return an empty matrix.

The `SetMarkerSize` method changes the marker sizes on a scale from 1 (the smallest) to 8 (the largest). This method is in the `Plot` class, so changing the size of markers in one graph does not change the size of markers in other graphs. (This is in contrast to the `DataObject` methods, which set the shape and color for all graphs that display an observation.) Finally, the `SetMarkerFillColor` method is used to make all markers hollow. Hollow markers can sometimes help reduce overplotting in scatter plots.

Figure 10.4 Changing Marker Shapes

In this example you used the `MARKER_X` and `MARKER_CIRCLE` shapes. The complete list of valid SAS/IML Studio marker shapes is given in [Table 10.2](#).

Table 10.2 Marker Shapes

<code>MARKER_SQUARE</code>	□
<code>MARKER_PLUS</code>	+
<code>MARKER_CIRCLE</code>	○
<code>MARKER_DIAMOND</code>	◇
<code>MARKER_X</code>	×
<code>MARKER_TRIANGLE</code>	△
<code>MARKER_INVTRIANGLE</code>	▽
<code>MARKER_STAR</code>	★

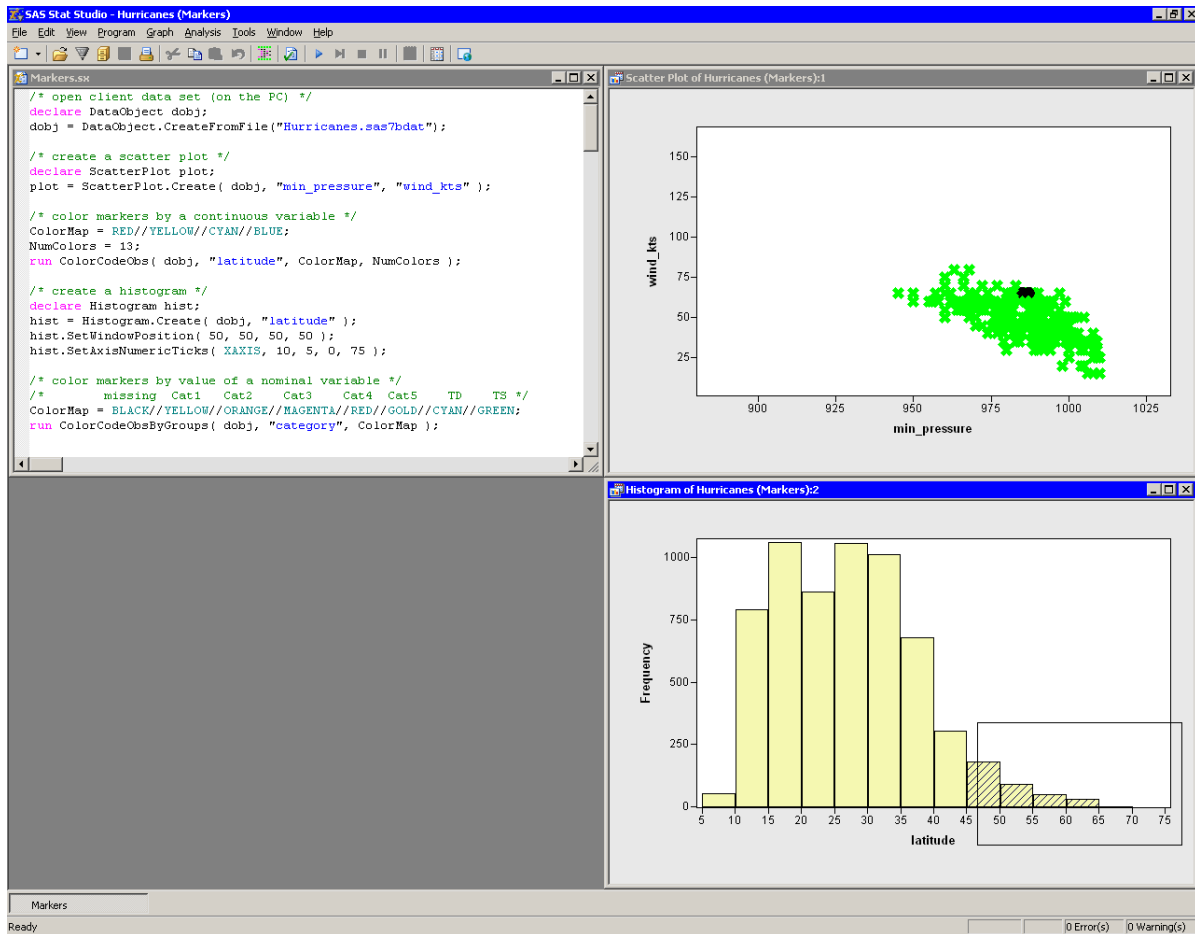
Show Only Selected Observations

A technique that is sometimes useful for exploring data is to show only observations that are selected. For example, suppose you are trying to understand how the `wind_kts` and `min_pressure` variables are distributed, given specific values for the `latitude` variable. Add the following statement at the bottom of the program window, and select **Program ► Run** from the main menu.

```
plot.ShowObs( false );
```

The scatter plot now displays only selected observations as shown in Figure 10.5. You can select bars in the histogram and examine how the wind speed and atmospheric pressure of storms vary as storms move from lower latitudes to higher latitudes. You can immediately see that storms at or above 45° tend to be weaker storms without well-developed eyes.

Figure 10.5 Showing Only Selected Observations



Chapter 11

Calling Functions in the R Language

Contents	
Introduction to Calling R Functions	71
Submit R Statements	72
Transfer between SAS and R Data Structures	73
Transfer from a SAS Source to an R Destination	73
Transfer from an R Source to a SAS Destination	74
Call an R Analysis from IMLPlus	74
Call R Packages and Graphics from IMLPlus	76

Introduction to Calling R Functions

R is a freely available language and environment for statistical computing and graphics. Like IMLPlus, the R language has features suitable for developers of statistical algorithms: the ability to manipulate matrices and vectors, a large number of built-in functions for computing statistical quantities and for creating statistical graphs, and the capability to extend the basic function library by writing user-defined functions. There are also a large number of R packages that implement specialized computations.

SAS/IML Studio has an interface to the R language that enables you to submit R statements from within your IMLPlus program. In previous chapters you learned how to transfer data between SAS/IML Studio and a SAS server, how to call SAS procedures, and how to read the results back into SAS/IML Studio. This chapter describes how to transfer data to R, how to call R functions, and how to transfer the results to a number of SAS data structures.

The program statements in this chapter are distributed with SAS/IML Studio. To open the program that contains the statements:

- 1 Select **File ►Open ►File** from the main menu.
- 2 Click **Go to Installation directory** near the bottom of the dialog box.
- 3 Navigate to the `Programs\Doc\STATGuide` folder.
- 4 Select the *R.sx* file.
- 5 Click **Open**.

In order to run the examples in this chapter, you must first install R on the same PC that runs SAS/IML Studio. For details on how to install R and which versions of R are supported, see the chapter “Accessing R” in the SAS/IML Studio online Help.

Submit R Statements

Submitting R statements is similar to submitting SAS statements. You use a SUBMIT statement, but add the R option: SUBMIT / R. All statements in the program prior to the next ENDSUBMIT statement are sent to R for execution.

The simplest program that calls R is one that does not transfer any data between the two environments. In the following program, SAS/IML is used to compute the product of a matrix and a vector. The result is printed. Then the SUBMIT statement with the R option is used to send an equivalent set of statements to R.

```
proc iml;
/* Comparison of matrix operations in IML and R */
print "----- SAS/IML Results -----";
x = 1:3;                                /* vector of sequence 1,2,3 */
m = {1 2 3, 4 5 6, 7 8 9};              /* 3x3 matrix */
q = m * t(x);                           /* matrix multiplication */
print q;

print "----- R Results -----";
submit / R;
  rx <- matrix( 1:3, nrow=1)              # vector of sequence 1,2,3
  rm <- matrix( 1:9, nrow=3, byrow=TRUE)  # 3x3 matrix
  rq <- rm %*% t(rx)                     # matrix multiplication
  print(rq)
endsubmit;
```

Figure 11.1 Output from SAS/IML and R

----- SAS/IML Results -----	
	<u>q</u>
	14
	32
	50
----- R Results -----	
	[,1]
[1,]	14
[2,]	32
[3,]	50

The printed output from R is automatically routed to the SAS/IML Studio output window, as shown in Figure 11.1. As expected, the result of the computation is the same in R as in SAS/IML.

Transfer between SAS and R Data Structures

Many research statisticians take advantage of special-purpose functions and packages written in the R language. To call an R function, the data must be accessible to R, either in a data frame or in an R matrix. This section describes how you can transfer data and statistical results (for example, fitted values or parameter estimates) between SAS and R data structures.

You can transfer data to and from the following SAS data structures:

- a SAS data set in a libref
- a SAS/IML matrix
- an IMLPlus DataObject

In addition, you can transfer data to and from the following R data structures:

- an R data frame
- an R matrix

Transfer from a SAS Source to an R Destination

The following table summarizes the frequently used methods that copy from a SAS source to an R destination. Several of these modules and methods are used in the program in the next section. For details of the transfer process and a full list of methods that transfer data, see the “Accessing R” chapter in the online Help.

Table 11.1 Transferring from a SAS Source to an R Destination

Method or Module	SAS Source	R Destination
ExportDataSetToR	SAS data set	R data frame
ExportMatrixToR	SAS/IML matrix	R matrix
DataObject.ExportToR	DataObject	R data frame

As a simple example, the following program transfers a data set from the Sashelp libref into an R data frame named `df`. The program then submits an R statement that displays the names of the variables in the data frame.

```
run ExportDataSetToR("Sashelp.Class", "df" );
submit / R;
  names(df);
endsubmit;
```

The R `names` function produces the output shown in Figure 11.2.

Figure 11.2 Sending Data to R

```
[1] "Name"    "Sex"     "Age"     "Height"  "Weight"
```

Transfer from an R Source to a SAS Destination

You can transfer data and results from R data frames or matrices to a SAS data set, a `DataObject`, or a SAS/IML matrix. The following table summarizes the frequently used methods that copy from an R source to a SAS destination.

Table 11.2 Transferring from an R Source to a SAS Destination

Method or Module	R Source	SAS Destination
<code>DataObject.AddVarFromR</code>	R expression	<code>DataObject</code> variable
<code>DataObject.CreateFromR</code>	R expression	<code>DataObject</code>
<code>ImportDataSetFromR</code>	R expression	SAS data set
<code>ImportMatrixFromR</code>	R expression	SAS/IML matrix

The next section includes an example of calling an R analysis. Some of the results from the analysis are then transferred into SAS/IML matrices and into variables in a `DataObject`.

The result of an R analysis can be a complicated structure. In order to transfer an R object via the previously mentioned methods and modules, the object must be coercible to a data frame. (The R object `m` can be coerced to a data frame provided that the function `as.data.frame(m)` succeeds.) There are many data structures that can not be coerced into data frames. As the example in the next section shows, you can use R statements to extract simpler objects and transfer the simpler objects.

Call an R Analysis from IMLPlus

The example in Chapter 4, “Calling SAS Procedures,” submits SAS statements to call the REG procedure. The example preforms a linear regression of the `wind_kts` variable by the `min_pressure` variable of the Hurricanes data. The following program repeats the same analysis, but does it by submitting statements to R:

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");
dobj.GetVarData( "wind_kts", w );
dobj.GetVarData( "min_pressure", p );
```

/* Step 1 */

```

/* send matrices to R */
run ExportMatrixToR( w, "Wind" );           /* Step 2 */
run ExportMatrixToR( p, "Pressure" );

print "----- In R -----";             /* Step 3 */
submit / R;
  Model    <- lm(Wind~Pressure, na.action="na.exclude")    # 3a
  ParamEst <- coef(Model)                                   # 3b
  Pred     <- fitted(Model)
  Resid    <- residuals(Model)
  print (ParamEst)                                         # 3c
endsubmit;

print "----- In SAS/IML -----";
run ImportMatrixFromR( pe, "ParamEst" );           /* Step 4 */
print pe[r={"Intercept" "min_pressure"}];

/* add variables to the DataObject */
dobj.AddVarFromR( "R_Pred", "Pred" );             /* Step 5 */
dobj.AddVarFromR( "R_Resid", "Resid" );
ScatterPlot.Create(dobj, "min_pressure", "R_Resid");

```

The output from this program is shown in [Figure 11.3](#). The program consists of the following steps:

1. The `GetVarData` method of the `DataObject` class copies the data for the `wind_kts` and `min_pressure` variables into SAS/IML vectors named `w` and `p`.
2. These vectors are sent to R by the `ExportMatrixToR` module. The names of the corresponding R vectors that contain the data are `Wind` and `Pressure`.
3. The `SUBMIT` statement with the `R` option is used to send statements to R. Note that comments in R begin with a hash mark (`#`, also called a number sign or a pound sign).
 - a) The `lm` function computes a linear model of `Wind` as a function of `Pressure`. The `na.action=` option specifies how the model handles missing values (which in R are represented by `NA`). In particular, the `na.exclude` option specifies that the `lm` function should not omit observations with missing values from residual and predicted values. This option makes it easier to merge the R results with the original data.
 - b) Various information is retrieved from the linear model and placed into R vectors named `ParamEst`, `Pred`, and `Resid`.
 - c) The parameter estimates are printed in R, as shown in [Figure 11.3](#).
4. The `ImportMatrixFromR` module transfers the `ParamEst` vector from R into a SAS/IML vector named `pe`. This vector is printed by the SAS/IML `PRINT` statement.
5. The `Pred` and `Resid` vectors are added to the `DataObject`. The new variables are given the names `R_Pred` and `R_Resid`. A scatter plot of the residual values versus the explanatory variable is created, similar to [Figure 6.1](#).

Figure 11.3 Calling an R Analysis

----- In R -----	
(Intercept)	Pressure
1333.354893	-1.291374

----- In SAS/IML -----	
pe	
Intercept	1333.3549
min_pressure	-1.291374

Note that you cannot directly transfer the contents of the **Model** object. Instead, various R functions are used to extract portions of the **Model** object, and those pieces are transferred.

As an alternative to steps 1 and 2, you can call the `ExportToR` method in the `DataObject` class. The `ExportToR` method writes an entire `DataObject` to an R data frame. For example, after creating the `DataObject` you could use the following statements to create an R data frame named `Hurr`:

```
dobj.ExportToR("Hurr");
submit / R;
  Model <- lm(wind_kts~min_pressure, data=Hurr, na.action="na.exclude")
endsubmit;
```

The R language is case-sensitive so you must use the correct case to refer to variables in a data frame.

The `SUBMIT` statement for R supports parameter substitution from SAS/IML matrices, just as it does for SAS statements. For example, you can substitute the names of analysis variables into a `SUBMIT` block by using the following statements:

```
YVar = "wind_kts";
XVar = "min_pressure";
submit XVar YVar / R;
  Model <- lm(&YVar ~ &XVar, data=Hurr, na.action="na.exclude")
  print (Model$call)
endsubmit;
```

Figure 11.4 shows the result of the `print (Model$call)` statement. The output shows that the values of the `YVar` and `XVar` matrices were substituted into the `SUBMIT` block.

Figure 11.4 Parameter Substitutions in a SUBMIT Block

```
lm(formula = wind_kts ~ min_pressure, data = Hurr, na.action = "na.exclude")
```

Call R Packages and Graphics from IMLPlus

You do not need to do anything special to call an R package. Provided that an R package is installed, you can call `library(package)` from inside a `SUBMIT` block to load the package. You can then call the functions in the package.

Similarly, you do not need to do anything special to call R graphics. The graph appears in the standard R graphics window.

The example in this section calls an R package and creates a graph in R.

In Chapter 6, “[Adding Curves to Graphs](#),” you called the KDE procedure to compute a kernel density estimate for the `min_pressure` variable in the Hurricanes data set. The following program reproduces that analysis by calling functions in the `KernSmooth` package and creating a histogram in R:

```
declare DataObject dobj;
dobj = DataObject.CreateFromFile("Hurricanes");
dobj.GetVarData("min_pressure", p);
run ExportMatrixToR( p, "Pressure" );

submit / R;
  library(KernSmooth)
  idx <-which(!is.na(Pressure))      # must exclude missing values (NA)
  p <- Pressure[idx]                #   from KernSmooth functions
  h = dpik(p)                       # Sheather-Jones plug-in bandwidth
  est <- bkde(p, bandwidth=h)       # est has 2 columns

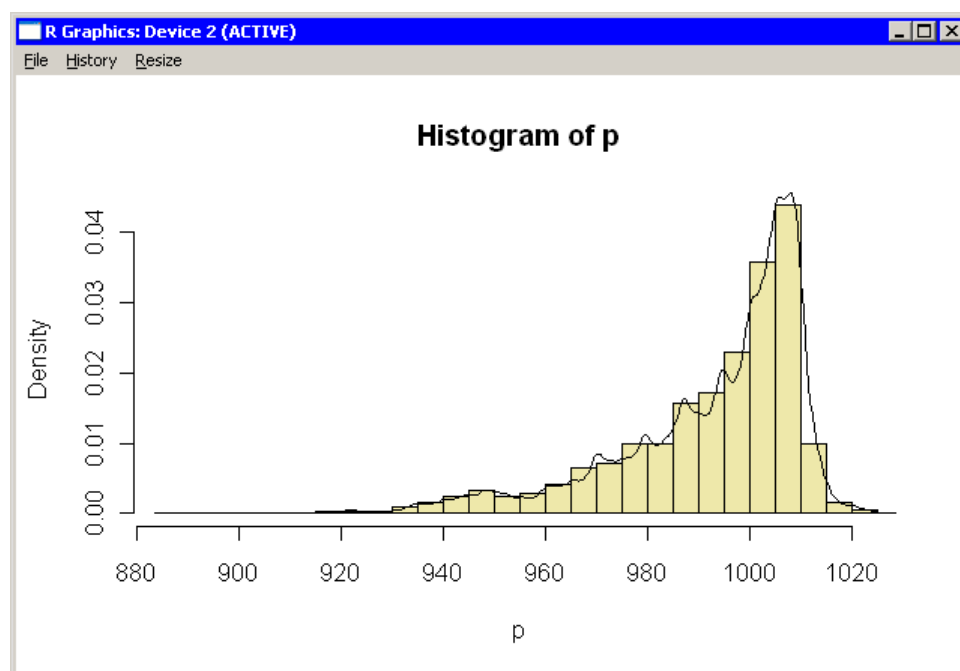
  hist(p, breaks="Scott", freq=FALSE, col="lightyellow") # histogram
  lines(est)                                           # kde overlay
endsubmit;
```

The program creates an R matrix `Pressure` from the data in the `min_pressure` variable. Because the functions in the `KernSmooth` package do not handle missing values, the nonmissing values in `Pressure` must be copied to a matrix `p`. The Sheather-Jones plug-in bandwidth is computed by calling the `dpik` function in the `KernSmooth` package. This bandwidth is used in the `bkde` function (in the same package) to compute a kernel density estimate.

The `hist` function creates a histogram of the data in the `p` matrix, and the `lines` function adds the kernel density estimate contained in the `est` matrix.

The R graphics window contains the histogram, which is shown in [Figure 11.5](#). You can compare the histogram and density estimate created by R with the IMLPlus graph shown in [Figure 6.4](#).

Figure 11.5 R Graphics



Index

- adding variables, 20
- AddVar method, 20
- AddVar.sx, 31
- AddVarFromR method, 74
- AddVars method, 20
- APPEND statement, 20
- AutoExec.sas, 14
- Axes.sx, 55
- axis view range, 57, 62

- base class, 24, 37

- case-sensitive, 6, 19
- class, 5
- client, 9, 11
- color map, 65
- ColorCodeObs module, 64
- ColorCodeObsByGroups module, 65
- CONCAT function, 51
- concatenation operator , 52
- continuous variables, *see* interval variables
- coordinate system, 37
- CopyServerDataToDataObject module, 18, 21, 32
- Create method, 20
- CREATE statement, 20
- CreateFromFile method, 17
- CreateFromR method, 74
- CreateFromServerDataSet method, 18
- creating a DataObject
 - from client data sets, 12, 17
 - from SAS/IML Matrices, 20
 - from server data sets, 14, 18

- data views, 8
- Data.sx, 17
- DataObject class, 6
 - purpose, 8
- DataObject methods
 - AddVar, 20
 - AddVarFromR, 74
 - AddVars, 20
 - Create, 20
 - CreateFromFile, 5, 17
 - CreateFromR, 74
 - CreateFromServerDataSet, 18
 - ExportToR, 73
 - GetVarData, 20, 37, 66
 - SetMarkerColor, 66
 - SetMarkerFillColor, 68
 - SetMarkerShape, 68
 - Sort, 6
 - WriteToFile, 18
 - WriteToServerDataSet, 19
 - WriteVarsToServerDataSet, 19
- DataTable class, 17
- DataView class, 24
- DataView methods
 - SetWindowPosition, 24, 60
- DELETE subroutines, 21
- discrete variables, *see* nominal variables
- dot notation, 5
- DrawInset module, 53
- DrawLegend module, 52
- DrawLine method, 37
- DrawSetPenColor method, 38
- DrawSetPenStyle method, 37
- DrawUseDataCoordinates method, 37
- dynamically linked, 8, 11, 23

- ENDSUBMIT statement, 19, 28
- ExportDataSetToR module, 73
- ExportMatrixToR module, 73
- ExportToR method, 73

- file extensions, 17, 18
- Fit.sx, 35

- GetPersonalFilesDirectory module, 18
- GetVarData method, 20, 37, 66
- Graphs.sx, 23

- help, *see* online Help
- Help ► Help Topics, 3
- histogram bin width, 60
- Histogram class, 24
- Histogram methods
 - Rebin, 62
- Hurricanes data set, 12

- IMLPlus
 - language, 1
 - programs, 4
- IMLPlus keywords
 - colors, 65
 - declare, 5
 - line styles, 37
 - marker shapes, 69
- ImportDataSetFromR module, 74

- ImportMatrixFromR module, 74
- in-memory data, 5, 6
- insets, 49, 53
- installation directory, 13
- interval variables, 63
- KDE procedure, 41
- legends, 49, 52
- LIBNAME statement, 14, 18
- library, 16
- librefs, 14
 - defining, 18
- line styles, 37
- LOC function, 66, 68
- markers
 - color, 63, 65
 - hollow, 68
 - predefined shape, 69
 - shape, 68
 - size, 68
- Markers.sx, 63
- methods, 5
- minor tick marks, 56
- missing values
 - color, 66
 - not colored, 65
- nominal variables, 65
- nonprinting columns, 46
- object-oriented, 5
- objects, 5
- OBS_ALL, 67
- ODS statement, 44
- ODS table name, 44
- ODS.sx, 43
- online Help, 3
- opening
 - client data set, 12, 17
 - server data set, 14, 18
- output data sets, 32
- Output Delivery System (ODS), 43
- output windows, 43
- personal files directory, 13, 18
- Plot methods
 - DrawLine, 37
 - DrawSetPenColor method, 38
 - DrawSetPenStyle method, 37
 - DrawUseDataCoordinates, 37
 - SetAxisLabel, 56
 - SetAxisMinorTicks, 56
 - SetAxisNumericTicks, 62
 - SetAxisTickAnchor, 56
 - SetAxisTickRange, 62
 - SetAxisTicks, 58
 - SetAxisTickUnit, 56
 - SetMarkerSize, 68
 - SetTitleText, 51
 - ShowAxisReferenceLines, 59
 - ShowTitle, 51
- polylines, 37
- predefined colors, 65
- Proc.sx, 27
- program windows, 5, 17
- programming language, 1
- programs
 - AddVar.sx, 31
 - Axes.sx, 55
 - Data.sx, 17
 - Fit.sx, 35
 - Graphs.sx, 23
 - Markers.sx, 63
 - ODS.sx, 43
 - Proc.sx, 27
 - R.sx, 71
 - Titles.sx, 49
- PUTN function, 51
- querying data, 7
- R language, 71
- R.sx, 71
- READ statement, 20, 41, 46
- ReBin method, 62
- REG procedure, 28, 32, 35
- Saffir-Simpson intensity scale, 58
- SAS library, 11
- saving
 - data to a SAS library, 16, 19
 - data to the client, 15, 18
- ScatterPlot class, 24
- selecting observations, 24
- selection rectangle, 24
- server, 9, 11
- server name, 14
- SetAxisLabel method, 56
- SetAxisMinorTicks method, 56
- SetAxisNumericTicks method, 62
- SetAxisTickAnchor method, 56
- SetAxisTickRange method, 62
- SetAxisTicks method, 58
- SetAxisTickUnit method, 56
- SetMarkerColor method, 66
- SetMarkerFillColor method, 68
- SetMarkerShape method, 68
- SetMarkerSize method, 68

- SetTitleText method, 51
- SetWindowPosition method, 24, 60
- show only selected observations, 69
- ShowAxisReferenceLines method, 59
- ShowTitle method, 51
- SORT call, 38
- SORT procedure, 41
- SUBMIT statement, 19, 28
 - parameter substitution, 29, 76
 - R statements, 72
- titles, 49, 51
- Titles.sx, 49
- unicode characters, vii
- USE statement, 20, 41, 46
- WriteToFile method, 18
- WriteToServerDataSet method, 19
- WriteVarsToServerDataSet method, 19

