



THE
POWER
TO KNOW.

SAS[®] Stat Studio **3.11** for SAS/STAT[®] Users



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *SAS® Stat Studio 3.11 for SAS/STAT® Users*. Cary, NC: SAS Institute Inc.

SAS® Stat Studio 3.11 for SAS/STAT® Users

Copyright © 2009, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-59994-941-3

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.

U.S. Government Restricted Rights Notice: Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987).

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

1st electronic book, February 2009

1st printing, March 2009

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/publishing 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 registered trademarks or trademarks of their respective companies.

Contents

Chapter 1. Introduction	1
Chapter 2. Reading and Writing Data	9
Chapter 3. Creating Dynamically Linked Graphics	21
Chapter 4. Calling SAS Procedures	25
Chapter 5. Adding Variables to the DataObject	29
Chapter 6. Adding Curves to Plots	31
Chapter 7. Reading ODS Tables	37
Chapter 8. Adding Titles, Legends, and Insets	41
Chapter 9. Adjusting Axes and Ticks	47
Chapter 10. Changing the Color and Shape of Observation Markers	55
Index	63

Release Notes

The following release notes pertain to SAS Stat Studio 3.11.

- Stat Studio requires SAS 9.2.
- This is an updated release of Stat Studio that enables access to remote SAS Workspace Servers.
- If you need to open a data set containing 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*.

Chapter 1

Introduction

What Is Stat Studio?

Stat Studio is a tool for data exploration and analysis. Stat Studio requires that you have a license for Base SAS, SAS/STAT, and SAS/IML. Stat Studio runs on a PC in the Microsoft Windows operating environment. You can use Stat Studio 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 Stat Studio interface with a logistic model for the probability of a passenger surviving 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 using only the Stat Studio graphical user interface (GUI). While the GUI provides powerful tools for analyzing data, you can also extend Stat Studio's built-in abilities by writing programs. Stat 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
- the data manipulation capabilities of Base SAS
- the dynamically linked graphics of Stat Studio

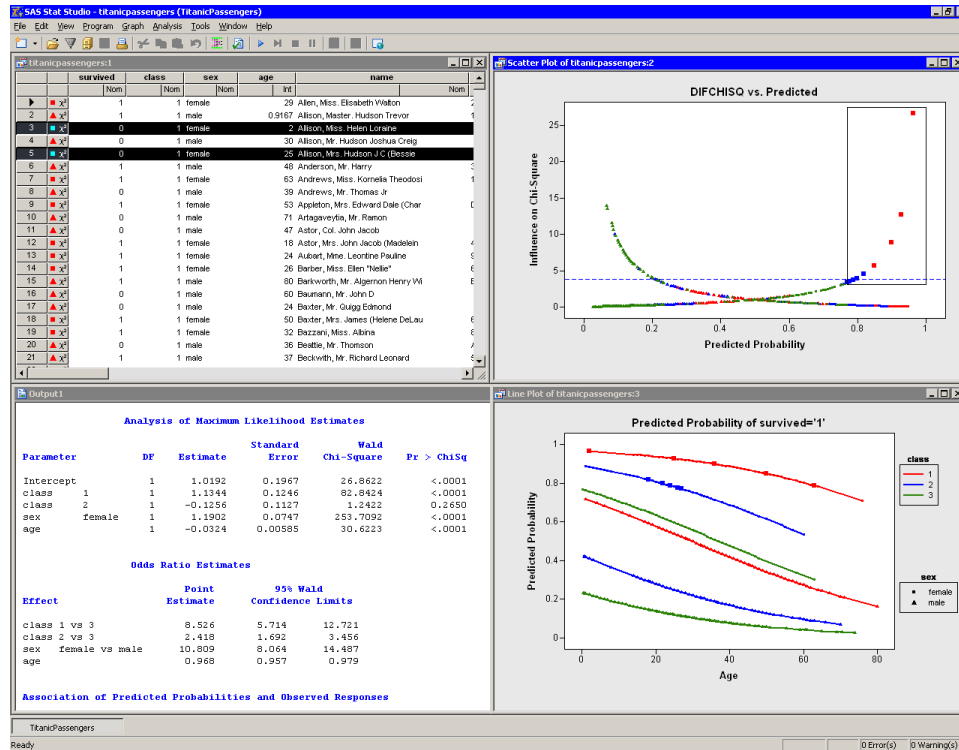


Figure 1.1. The Stat Studio Interface

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

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

The Purpose of This Book

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

In particular, this book focuses on how to create dynamically linked graphics 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 graphics 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 Stat Studio. You can learn how to use the Stat Studio GUI to conduct exploratory data analysis and standard statistical analyses in *Stat Studio User's Guide*. That book also shows you how to perform many of the tasks in this chapter by using menus in the Stat Studio GUI. You can learn more advanced programming commands and techniques from the Stat Studio online Help, which you can display by selecting **Help ► Help Topics** from the main menu.

Why Program in Stat Studio?

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

- create your own customized statistical graphics
- add legends, curves, maps, or other custom features to statistical graphics
- 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 IML
- share analyses with colleagues who also use Stat Studio
- 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 predicting mortality as a function of a patient's age and the severity of her condition. The program uses IML 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 IML to compute adjusted mortality rates. It presents Stat Studio's dynamically linked graphics 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.

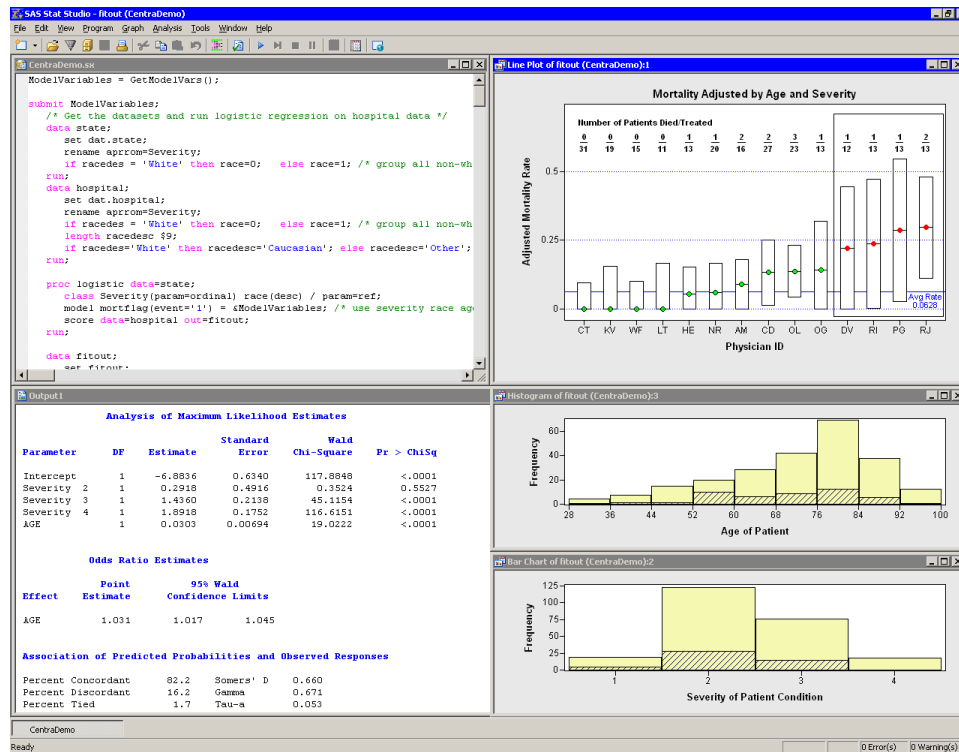


Figure 1.2. Results of an IMLPlus Program

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](#), [Chapter 7](#)) and, optionally, use IML for additional analysis.
4. Create graphs that use the results ([Chapter 3](#), [Chapter 5](#), [Chapter 6](#)).
5. Customize attributes of the graphs ([Chapter 8](#), [Chapter 9](#), [Chapter 10](#)).

The chapters of this book describe these steps in detail. Later chapters build on earlier chapters.

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 Stat Studio menu. For your convenience, the program for each chapter is also distributed with Stat 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 graphics. The section “[Understanding Classes, Objects, and Methods](#)” on page 5 explains these concepts in general, while the section “[The DataObject Class](#)” on page 6 focuses on a class that is particularly important in Stat 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 8 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 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 that query, retrieve, or manipulate the data. These functions are usually called *methods*.

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 SAS/IML, all variables are matrices. In IMLPlus, a variable is implicitly assumed to be an 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 specify the data to 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.

Note: 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.”

Caution: IML is not a case-sensitive language. 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 Stat 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`.

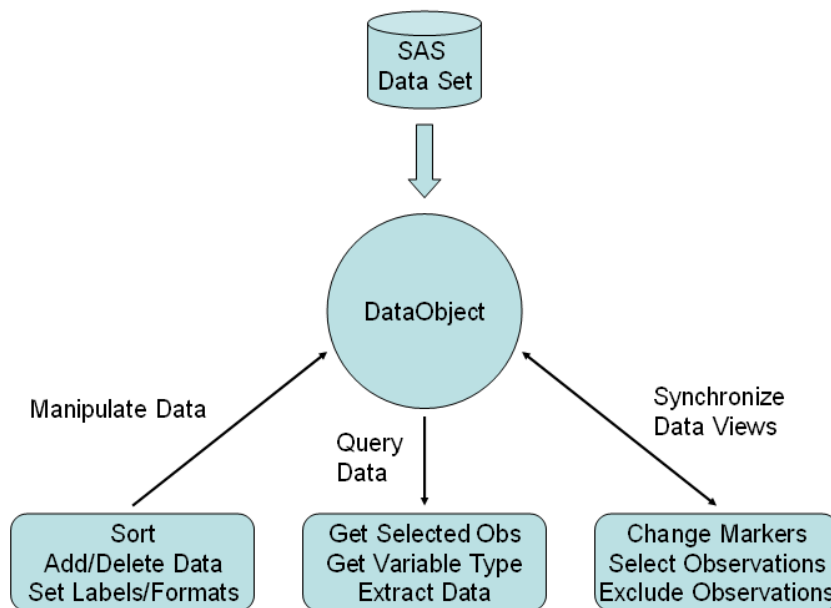


Figure 1.3. Using a `DataObject`

A `DataObject` is usually created from a SAS data set. (Other methods of creating `DataObjects`, such as from Excel files or from IML matrices, are discussed in the online Help.) However, once 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 using 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 containing 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](#).

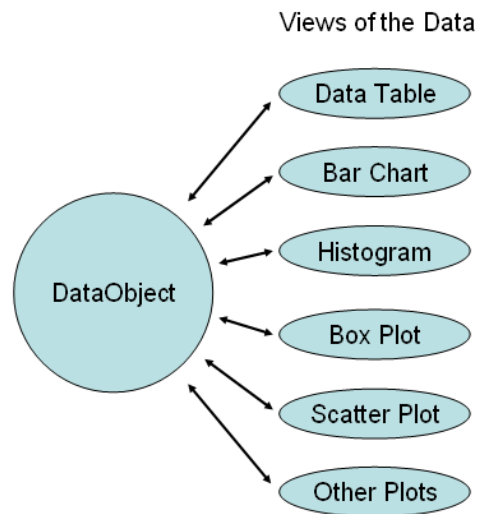


Figure 1.4. The `DataObject` Role

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 will have 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?

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

There is a fundamental difference between the Stat Studio graphics and the Stat Studio analyses. The `DataObject` class, which coordinates all of the dynamically linked graphics and tables, runs on the client and keeps its data in memory on the client. Similarly, the graphics 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.

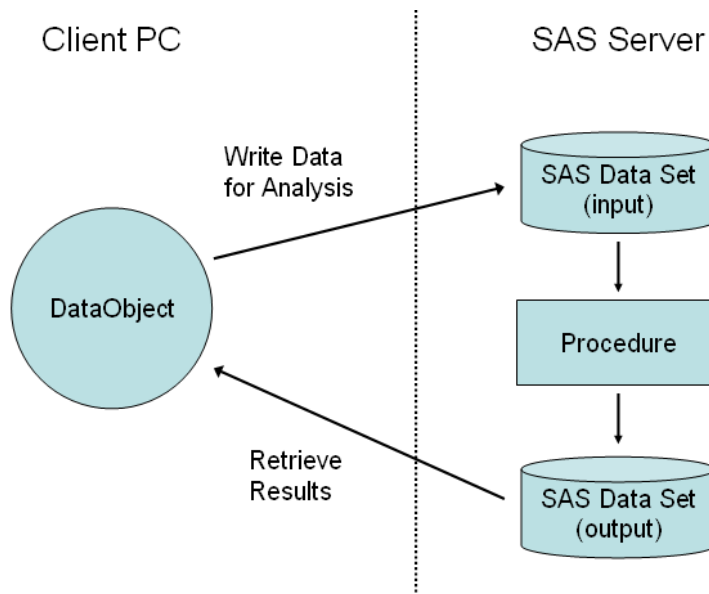


Figure 1.5. Data Flow

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

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

Dynamically linked graphics require an in-memory DataObject running on the client PC. Calling a SAS procedure requires a SAS data set in a library on the server. Therefore, if you are exploring data by using graphics 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 Stat 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.

Using 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 running Stat Studio. 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\Stat Studio\3.1\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 defined by using the LIBNAME statement. For example, the following are server data sets:

- SASHELP.CLASS
- WORK.data1
- mylib.research

Opening Client Data Sets

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 on the **Data Sets** folder.
4. Select the **Hurricanes.sas7bdat** file.
5. Click **Open**.

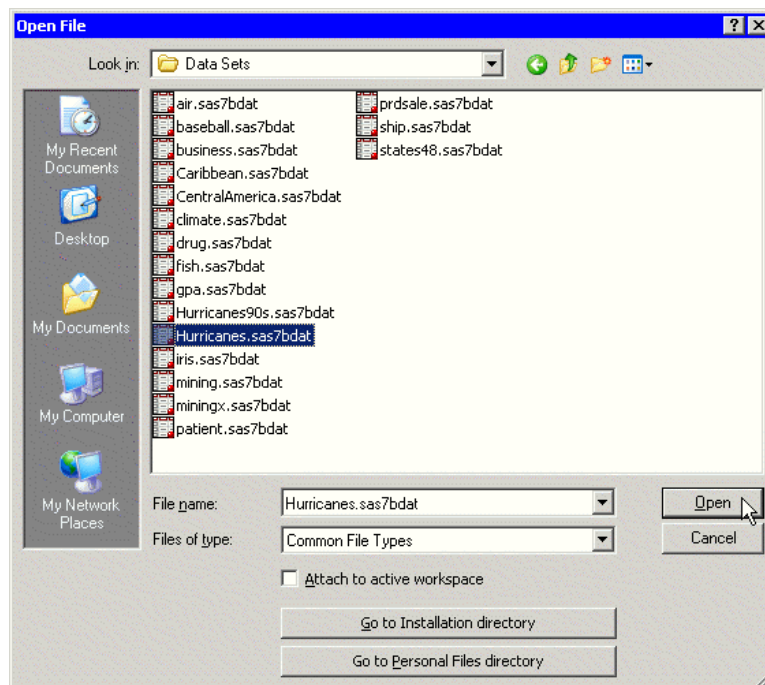


Figure 2.1. Opening a Client Data Set

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\userid\My Documents\My Stat Studio Files
Windows Vista	C:\Users\userid\Documents\My Stat Studio Files

Opening Server Data Sets

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 on the node labeled with your server name to open it. If SAS is running on your PC, the server name is **My SAS Server**.
3. Click on the **SASHELP** folder to view the data sets in the **SASHELP** library.
4. Select the **CLASS** data set.
5. Click **OK**.

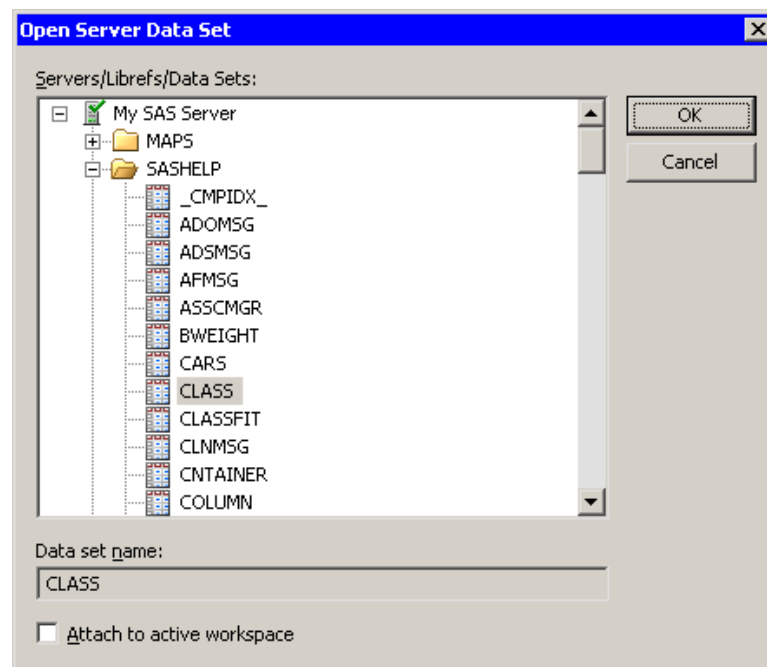


Figure 2.2. Opening a Server Data Set

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, SAS 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.

Using the GUI to Write a SAS Data Set

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

Saving Data to the Client

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

1. Activate the data table by clicking on 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**.

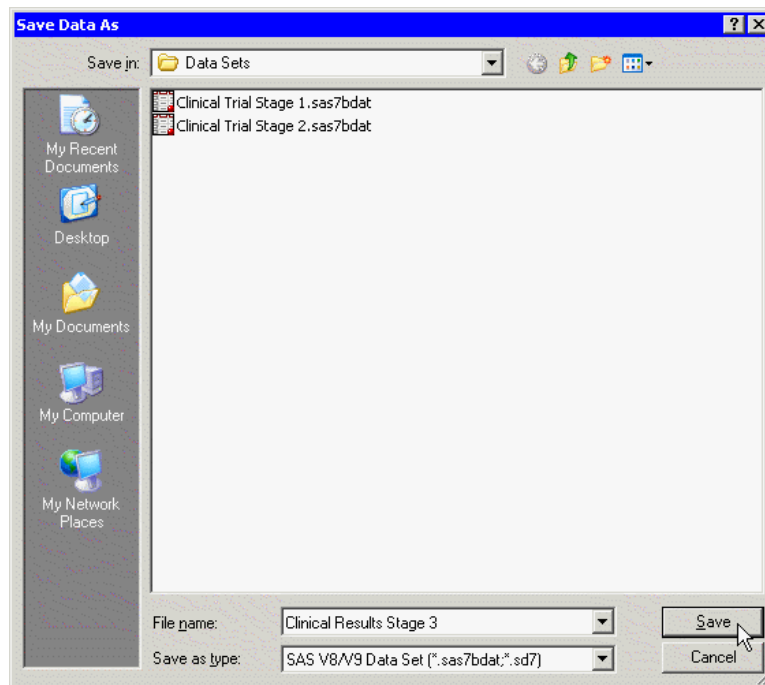


Figure 2.3. Saving to a Client Data Set

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. Stat Studio provides easy navigation for loading files in this directory. Furthermore, the section “[Opening Client Data Sets](#)” on page 14 explains that this directory is automatically searched when you use a program to create a DataObject from a data set.

Saving Data to a SAS Library

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

1. Activate the data table by clicking on 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 on a node to see the available libraries for a server. If SAS is running on your PC, the server name is **My SAS Server**.
4. Click on the node for a library. Each Stat Studio workspace has its own private **WORK** library, but other libraries (such as **SASUSER**) are shared across all Stat Studio workspaces.
5. Type a valid SAS data set name in the **Data set name** field.
6. Click **OK**.

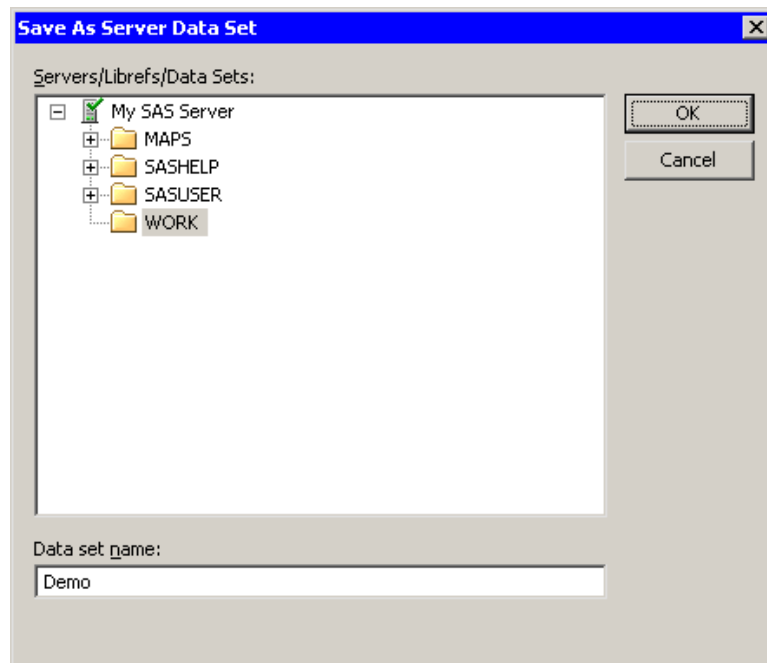


Figure 2.4. Saving to a SAS Library

Using a Program to Read a SAS Data Set

As explained in [Chapter 1, “Introduction,”](#) 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 Stat Studio. To open the program containing 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**.

Opening Client Data Sets

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 Stat Studio online Help.)

If you omit the file extension from the argument to `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 step 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 to the `CreateFromFile` method. If, however, you specify a partial pathname as in the preceding example, then Stat Studio searches for the file relative to certain directories. The `Hurricanes` data set is distributed with Stat Studio. The directory containing Stat Studio sample data sets (`C:\Program Files\SAS\Stat Studio\3.1\Data Sets`) is, by default, one of the directories automatically searched. See the Stat Studio online Help for information about how to add or change directory search paths.

Opening Server Data Sets

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 “Using IML Matrices to Store Data” on page 17 and Chapter 5, “Adding Variables to the `DataObject`.”

Using 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.

Saving Data to the Client

If you want to save data in a `DataObject` to a `sas7bdat` 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 saying that the file format is not supported.

It is often convenient to save data to your personal files directory, as described in the “Opening Client Data Sets” section on page 10. 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 );
```

Saving 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 to 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.

Caution: Java is a case sensitive language, so the last argument to 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.

Viewing 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 `PROC PRINT` or `PROC CONTENTS` 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 SAS 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 Stat Studio as explained in the section “[Opening Server Data Sets](#)” on page 11.

Using IML Matrices to Store Data

Although this book does not require previous knowledge of SAS/IML, experienced users of IML might wonder how to transfer data between IML 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 IML matrices. The USE statement opens a SAS data set, and the READ statement reads server data into IML matrices. IML also has a CREATE statement that creates a server data set and an APPEND statement that writes variables from IML matrices. These statements are documented fully in the *SAS/IML User’s Guide*.

Creating a DataObject from IML Matrices

If you have data in 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 Stat 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.

Adding 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 Stat 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.

Copying Variables to IML Matrices

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

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

The result is a 5×1 IML matrix **x** that contains the values of the **XVar** variable. You can name the IML 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 IML 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 2.2–2.4 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
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
IML matrix	DataObject.GetVarData

Table 2.4. Reading and Writing Server Data in IML

Action	Statement
Read from server into IML	USE, READ
Write server data set from IML	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 IML `DELETE` subroutine.

Chapter 3

Creating Dynamically Linked Graphics

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

[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 graphics to visualize your data and relationships between variables.

The program statements in this chapter are distributed with Stat Studio. To open the program containing 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**.

Creating 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 Stat 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 Stat Studio online Help.

Selecting 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](#).

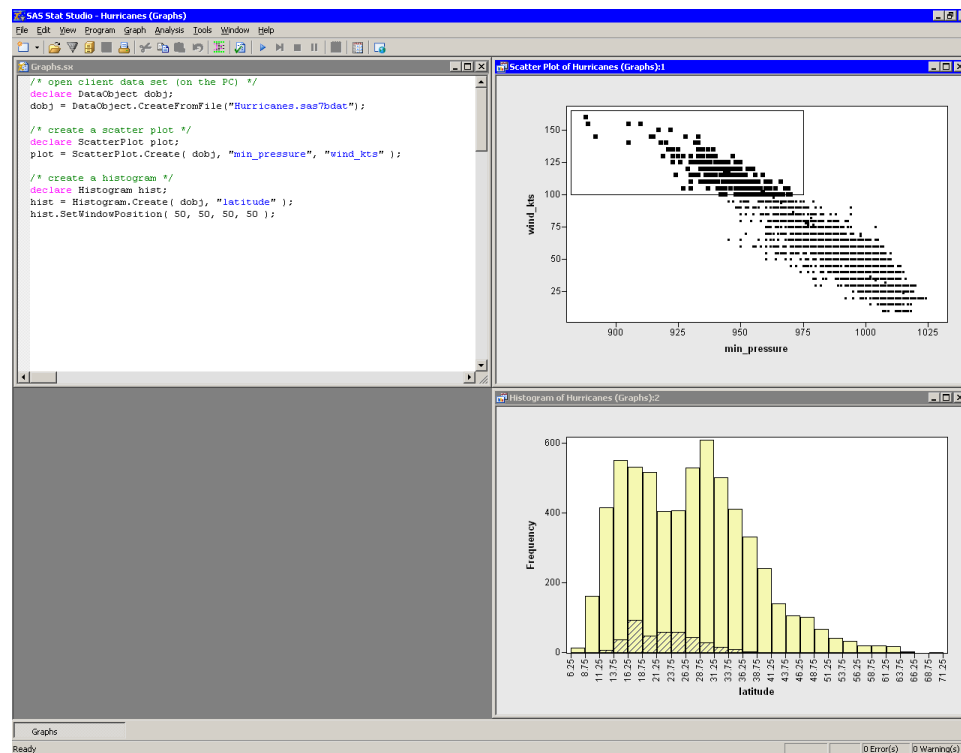


Figure 3.1. Creating Graphics

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.

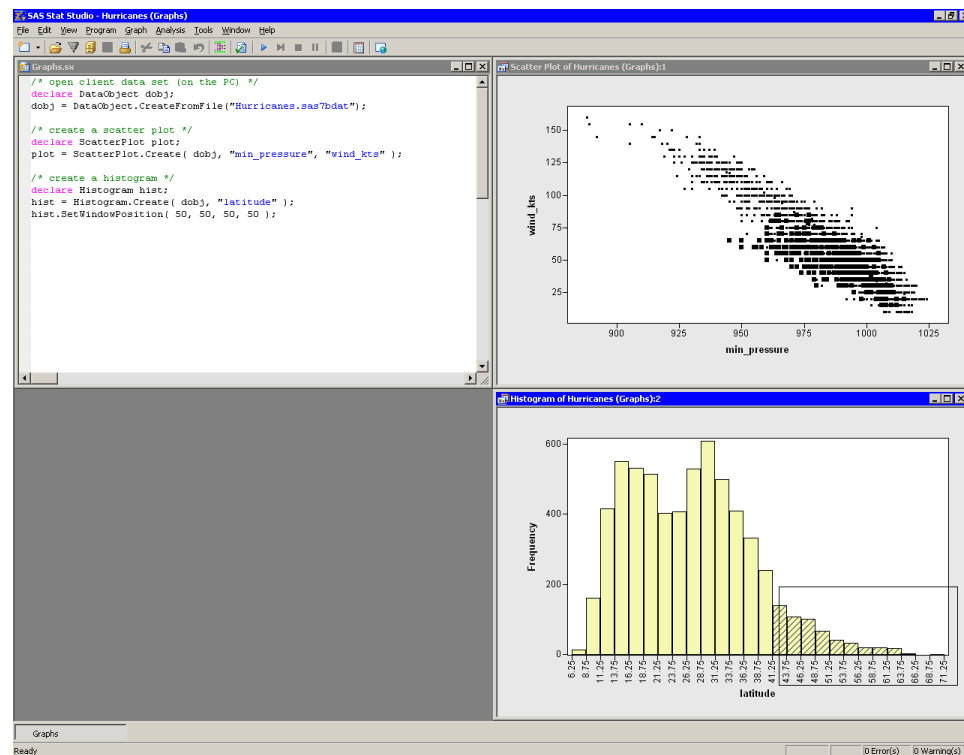


Figure 3.2. Storms in Relatively Northern Latitudes

Summary of Creating Graphics

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

Chapter 4

Calling SAS Procedures

In previous chapters you learned how to open a data set and how to explore the data by creating dynamically linked graphics. Qualitatively exploring your data might lead you to formulate statistical models. SAS procedures such as those in Base SAS and SAS/STAT can help you 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 Stat Studio. To open the program containing 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**.

Creating a Data Set for the Procedure

Recall from [Chapter 1, “Introduction,”](#) that the DataObject, which coordinates all of the dynamically linked graphics and tables, runs on the client and keeps its data in memory on the client. SAS procedures, however, 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 Graphics,”](#) 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.

Calling a Procedure

Now that there is a SAS data set on the server containing variables of interest, 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 SAS 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, Stat Studio calls the REG procedure with the CORR option. The CORR option prints out the correlation matrix for the variables in the MODEL statement.

The REG procedure displays its tables in the Stat 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 $\text{wind_kts} = 1333 - 1.291 \times \text{min_pressure}$.

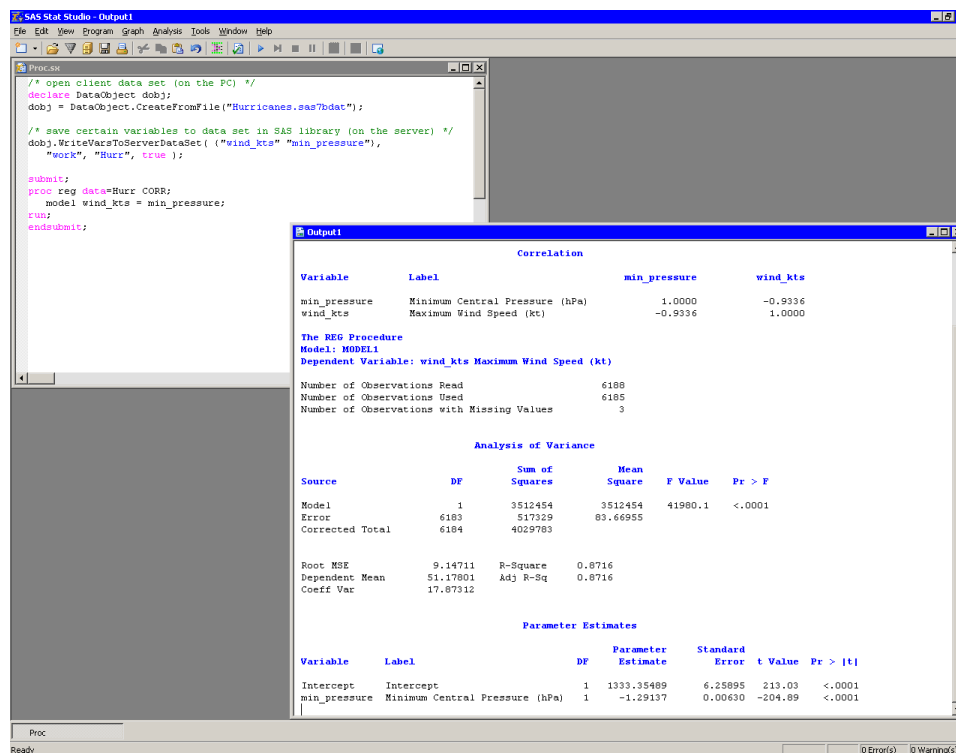


Figure 4.1. Calling the REG Procedure

Chapter 5

Adding Variables to the 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 Stat Studio. To open the program containing 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**.

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 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 on 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;  
    model wind_kts = min_pressure;  
    output out=RegOut R=Residual;  
run;  
endsubmit;
```

When you run the program, Stat 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`.

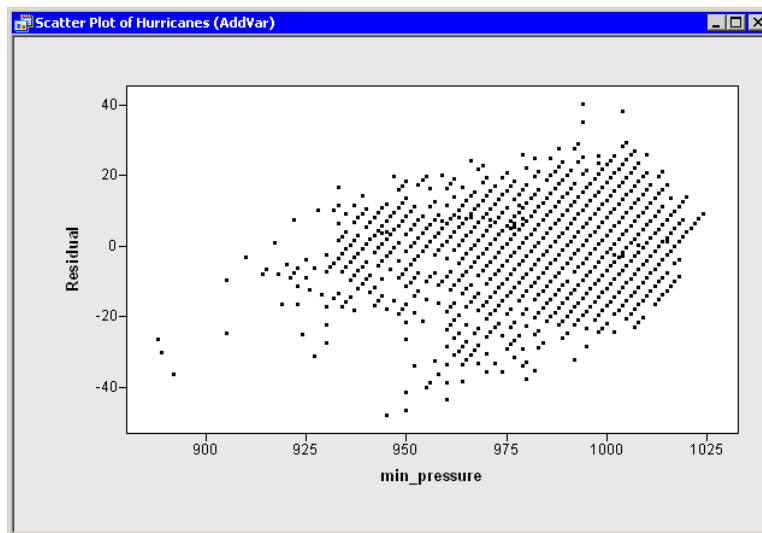


Figure 5.1. Creating a Residual Plot

Chapter 6

Adding Curves to Plots

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 Stat Studio. To open the program containing 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**.

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.

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

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

submit;
proc reg data=Hurr;
    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 Stat Studio online Help.

Drawing 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 $\text{Residual} = 0$.

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.

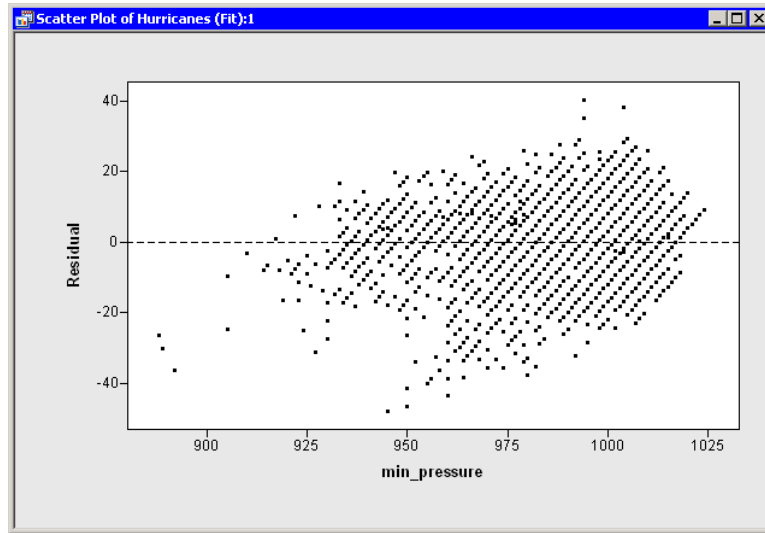


Figure 6.1. A Plot with a Reference Line

Drawing 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 plotting 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 an IML matrix containing 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 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.

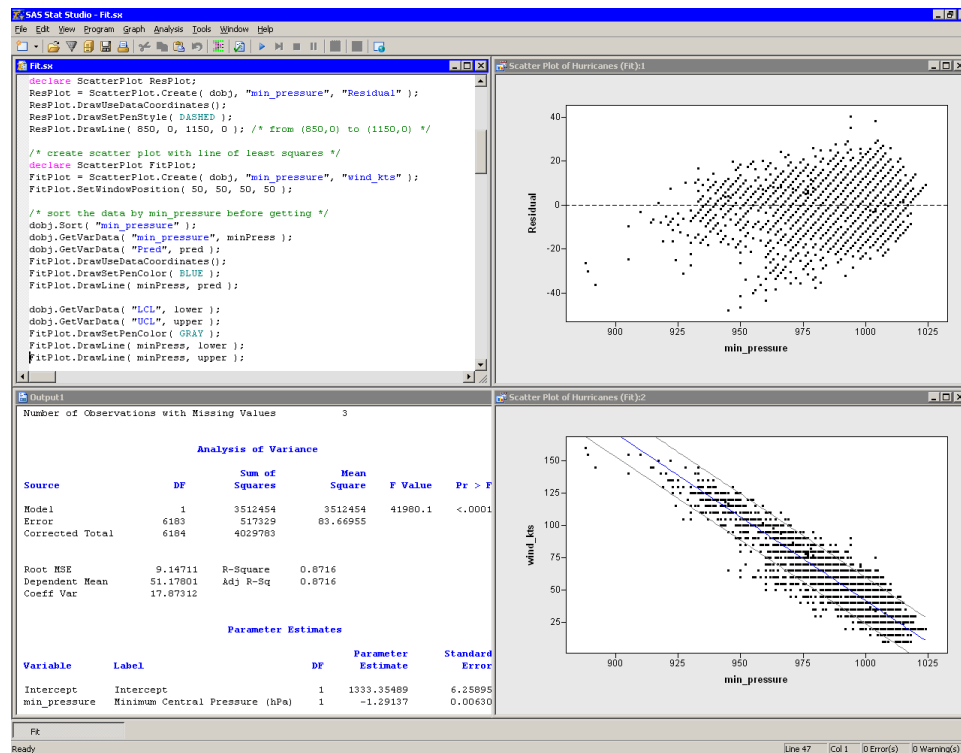


Figure 6.2. A Fitted Model and Confidence Intervals

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 advantage of reading variables into the `DataObject`

is that you can open a data table to see the values of the added variables, and you can use these variables in subsequent analyses. Furthermore, you can use the dynamically linked graphics 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.

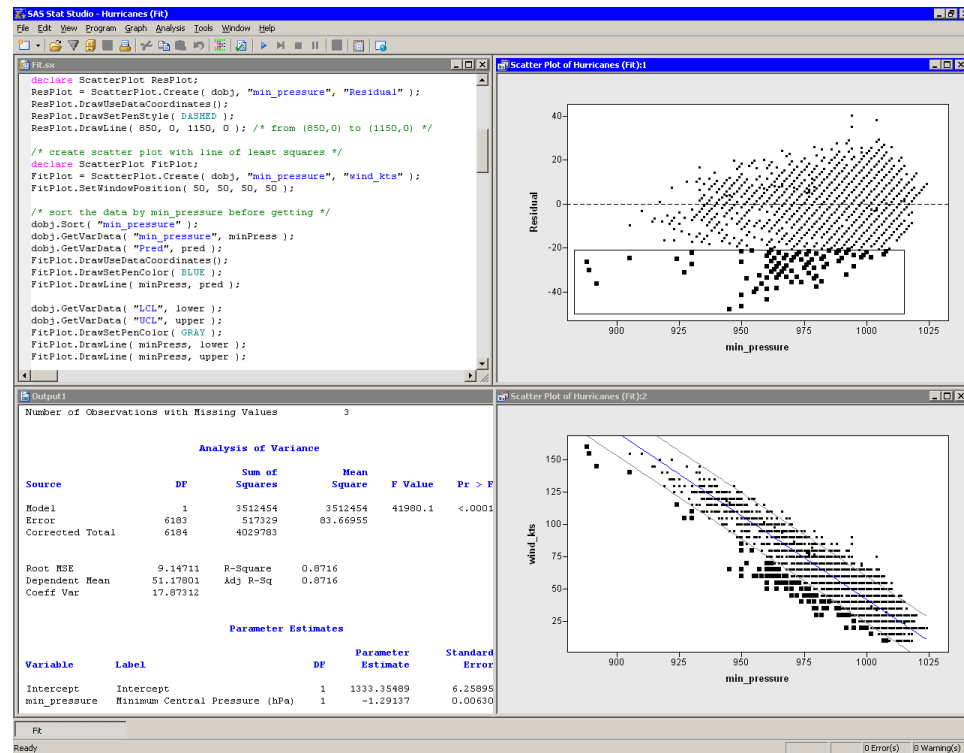


Figure 6.3. Observations with Large Negative Residuals

Drawing a Curve from Variables in a Data Set

Sometimes the output of a procedure contains a different number of observations than your data contain. This eliminates the possibility of reading the output into the DataObject containing 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 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;
  run;
endsubmit;

```

The KDE procedure creates the data set `work.KDEOut` on the SAS server containing 401 observations. The data set contains a variable `Value` consisting 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.

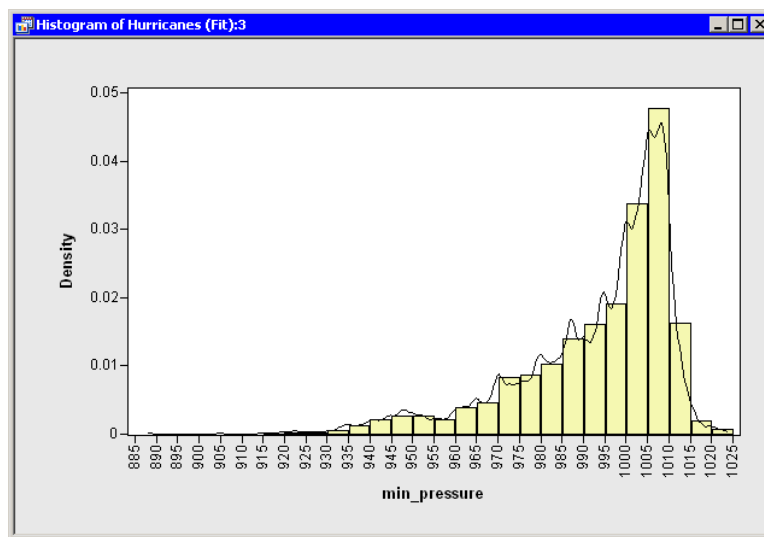


Figure 6.4. A Histogram Overlaid with a Kernel Density Estimate

Chapter 7

Reading ODS Tables

In [Chapter 6, “Adding Curves to Plots,”](#) you generated output data sets by using SAS procedures and learned how to read variables from the server into the DataObject and into 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 Stat Studio. To open the program containing 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**.

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 Stat Studio is called the *output windows*. 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 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 on 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.

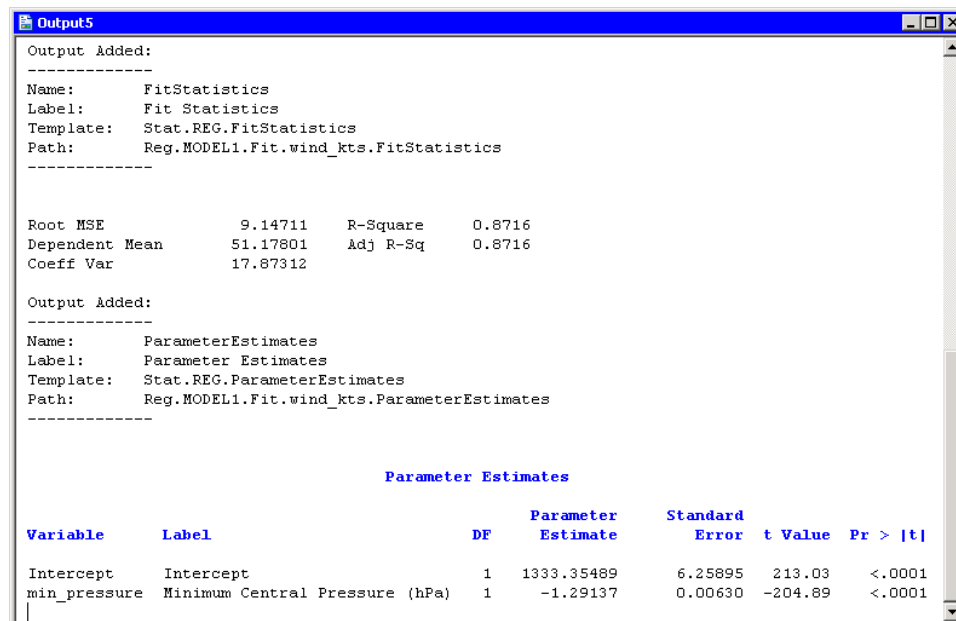
```

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;
    model wind_kts = min_pressure;
run;
ods trace off;
endsubmit;

```



The screenshot shows the SAS Output window titled 'Output5'. It displays the output of the ODS TRACE ON statement, listing the names of the tables generated by the PROC REG procedure. The tables listed are FitStatistics and ParameterEstimates. The FitStatistics table contains summary statistics for the regression model, including Root MSE, Dependent Mean, Coeff Var, R-Square, and Adj R-Sq. The ParameterEstimates table contains the parameter estimates for the regression model, including the Intercept and the slope for min_pressure. The output is formatted as a table with columns for Variable, Label, DF, Parameter Estimate, Standard Error, t Value, and Pr > |t|.

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

Figure 7.1. Printing ODS Tables Names

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 output window 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;
  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 on the **REGNOBS** data set.
5. Click **OK**.
6. Repeat the previous steps to open the **REGFITSTAT** and **REGPARAMEST** data sets.

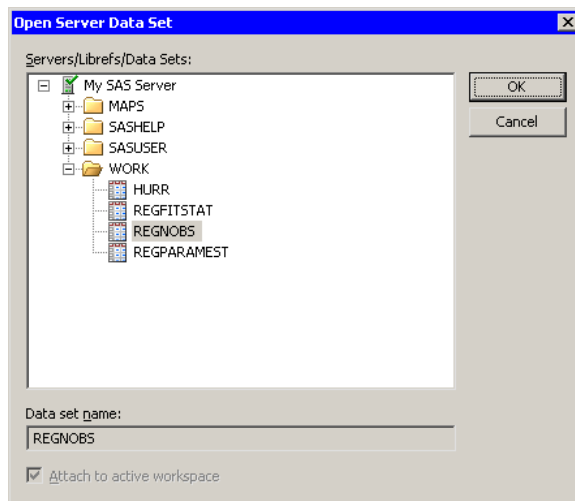


Figure 7.2. Opening a Server Data Set

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 containing the information that you want to read into IML. 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 IML matrices. The USE statement opens a SAS data set, and the READ statement reads server data into IML 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 "-----";
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 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.

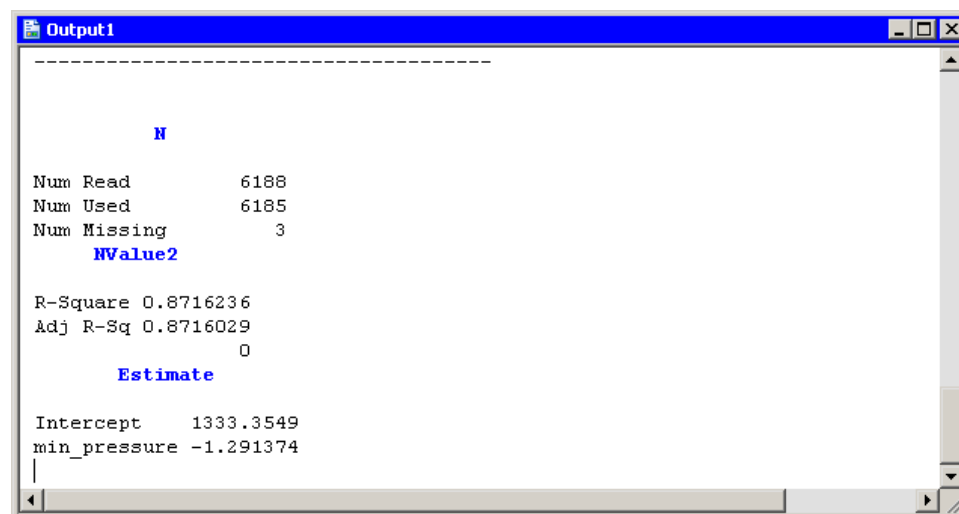


Figure 7.3. Creating Matrices from ODS Tables

Chapter 8

Adding Titles, Legends, and Insets

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 IML matrices. This chapter shows you how to create titles, legends, and *insets* on plots. (An inset is a box drawn on the plot containing statistics.) In particular, you learn how to incorporate the information from ODS tables into titles, legends, and insets.

The program statements in this chapter are distributed with Stat Studio. To open the program containing 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**.

Suppose you perform a linear regression with PROC REG, and you create a scatter plot showing the fitted model and 95% confidence intervals for individual predictions as shown in [Figure 8.1](#). You learned how to do this in [Chapter 6, “Adding Curves to Plots.”](#)

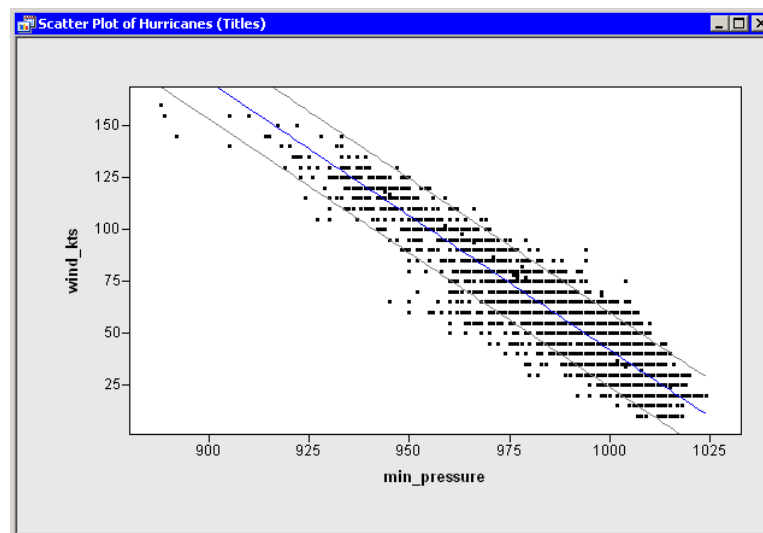


Figure 8.1. A Fitted Model and Confidence Intervals

Suppose further that you want the plot to include the following information about the linear regression:

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


You can add a title to a plot 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 function `PUTN`. You can then concatenate pieces of the title together using the 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();
```

Adding a legend is usually accomplished by using the `DrawLegend` module that is distributed with Stat Studio. The module is documented in the Stat 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 IML concatenation operator `||`. This operator takes the two predefined integers `BLUE` and `GRAY` and concatenates them into a 1×2 matrix.

Caution: Trying to form the matrix by using the statement

```

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

```

does not work because IML interprets that statement as forming a character matrix with values “BLUE” and “GRAY.” 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! */

```

You can add an inset by using the `DrawInset` module that is distributed with Stat Studio. The module is documented in the Stat Studio online Help. The code that follows shows 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. [Figure 8.2](#) shows how the inset appears.

```

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 );

```

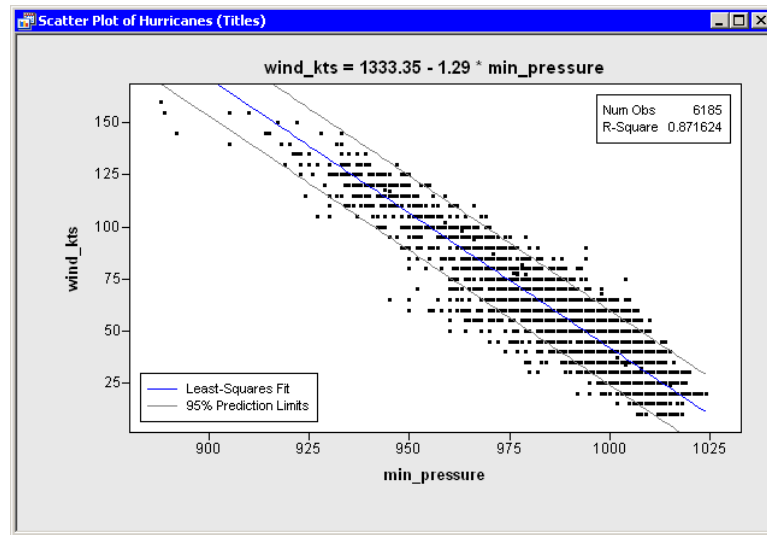


Figure 8.2. Plot with Title, Legend, and Inset

Chapter 9

Adjusting Axes and Ticks

In [Chapter 3, “Creating Dynamically Linked Graphics,”](#) you created standard statistical graphics. 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 Stat Studio. To open the program containing 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**.

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. Stat 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.

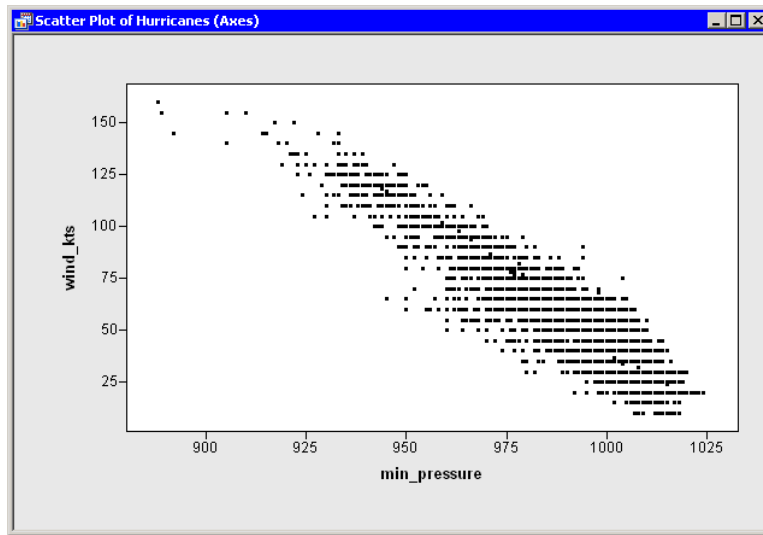


Figure 9.1. Default Axes

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 Stat 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 );
```

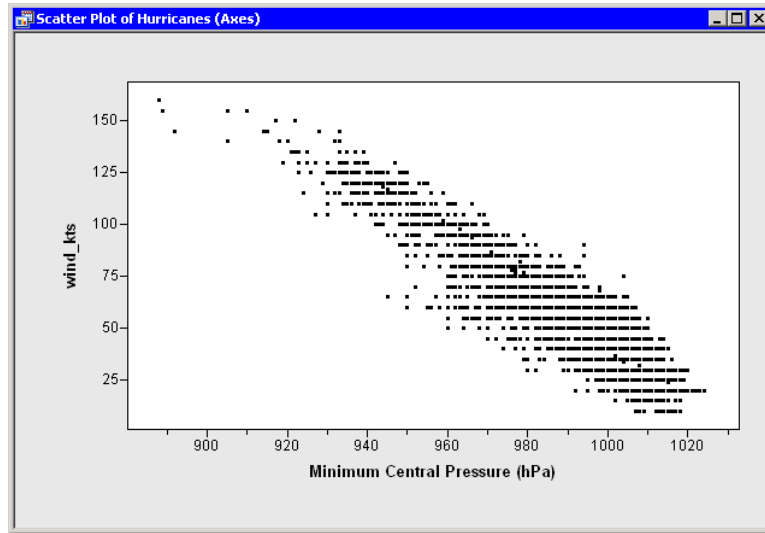


Figure 9.2. New Horizontal Axis

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 );
```

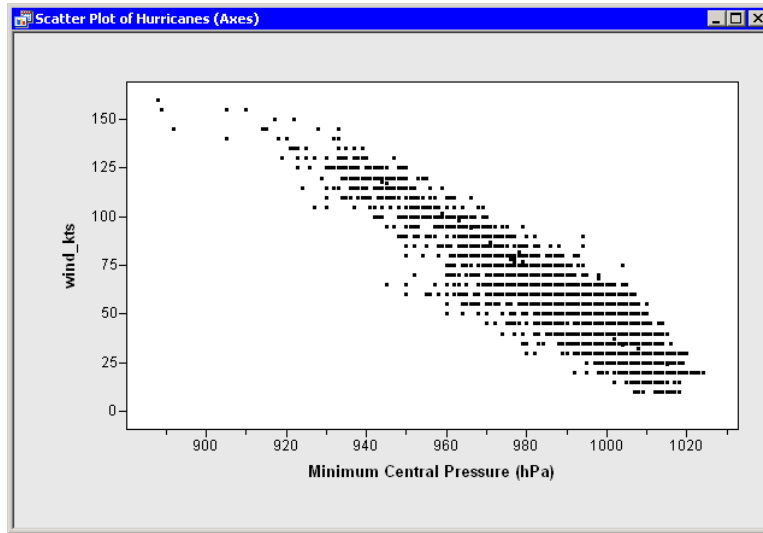


Figure 9.3. Vertical Axis Range That Includes Zero

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 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 );
```

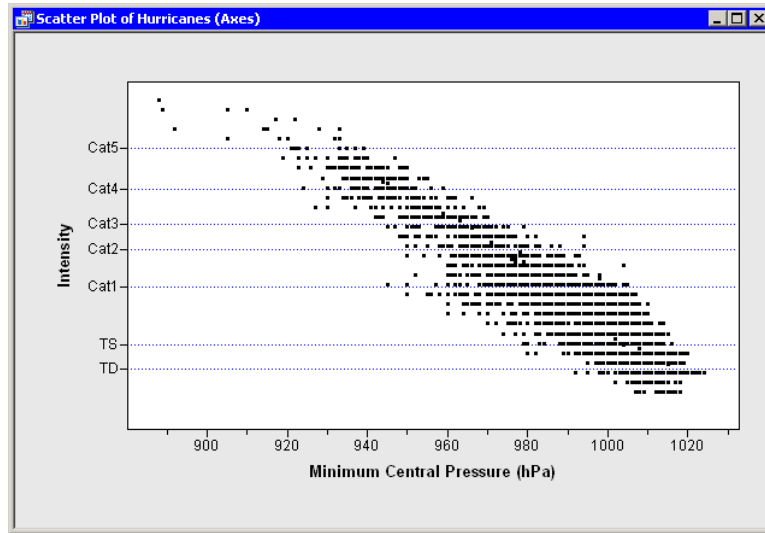



Figure 9.4. Vertical Axis with Custom Ticks

Adjusting Tick Marks for Histograms

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 );
```

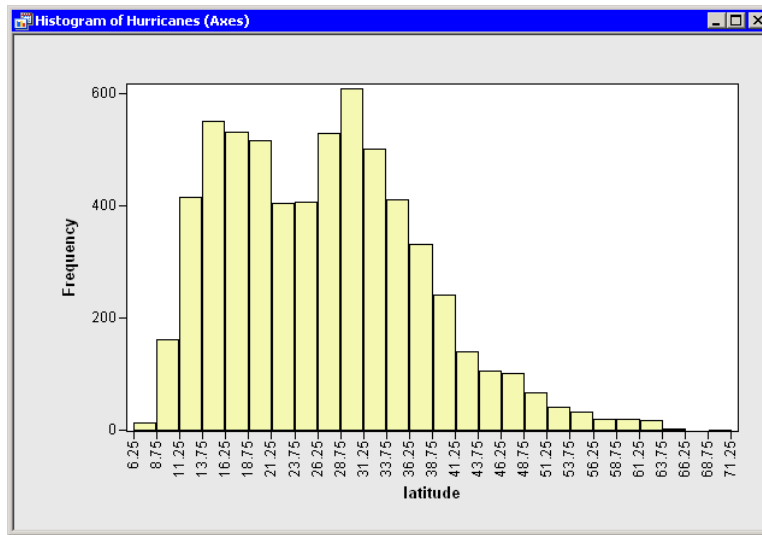


Figure 9.5. Histogram with Default Bins

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 Stat 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](#).

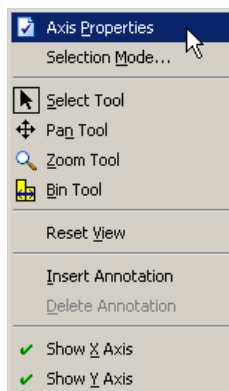


Figure 9.6. Plot Area Pop-up Menu

2. Select **Axis Properties** from the pop-up menu. The Axis Properties dialog box appears as in [Figure 9.7](#). Note that 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**.

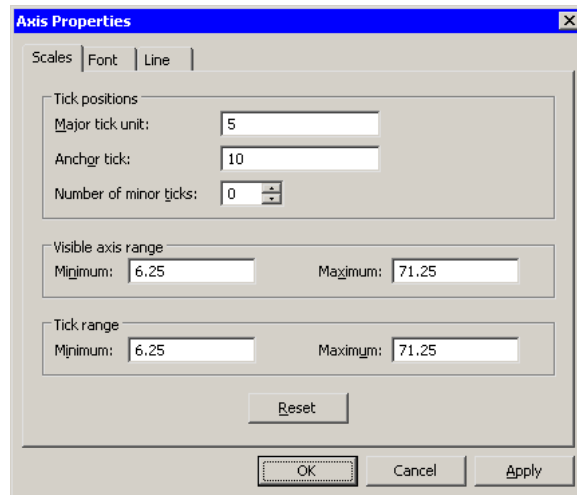


Figure 9.7. Specifying Histogram Bins

The histogram updates to reflect the new histogram bin locations ([Figure 9.8](#)). Note that 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.

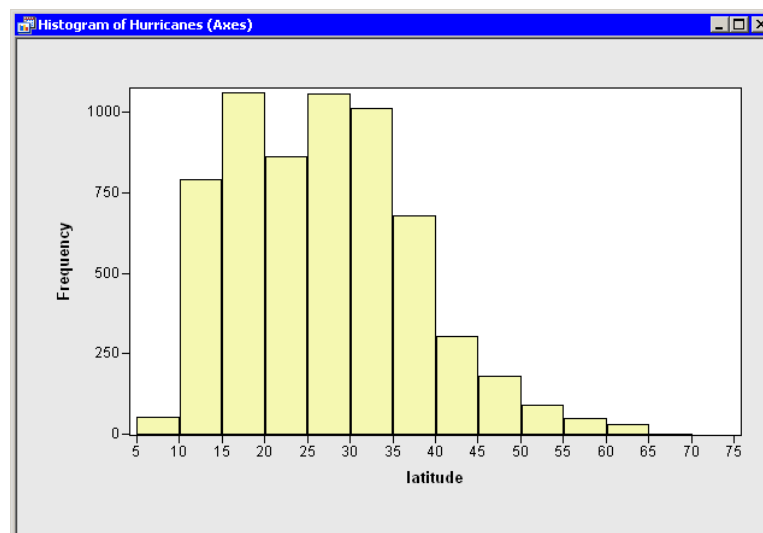


Figure 9.8. Histogram with Customized Bins

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 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

In [Chapter 3, “Creating Dynamically Linked Graphics,”](#) you created standard statistical graphics. 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 Stat Studio. To open the program containing 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**.

Using Interval Variable Values to Color Markers

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 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 Stat 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 Graphics.”](#) They also adjust the histogram bins as described in [Chapter 9, “Adjusting Axes and Ticks.”](#) The resulting scatter plot is shown in [Figure 10.1](#).

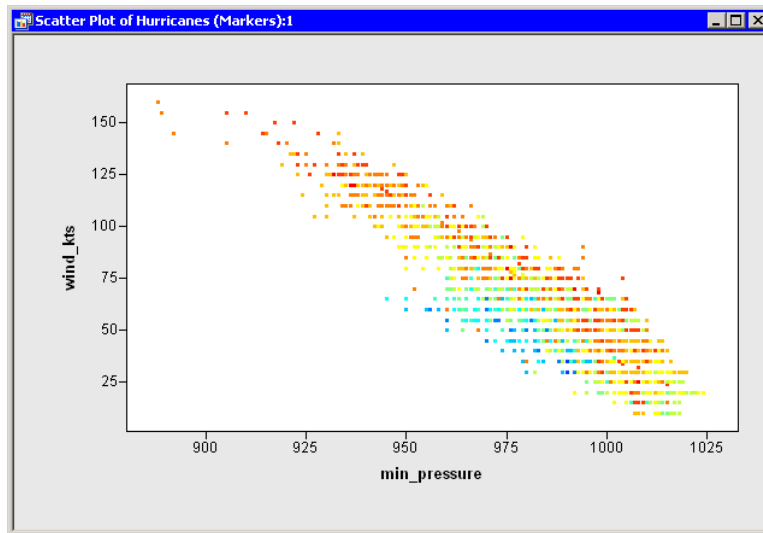


Figure 10.1. Markers Colored by an Interval Variable

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 Stat Studio, or you can create your own colors by specifying their RGB or hexadecimal values as described in the Stat Studio online Help. [Table 10.1](#) lists the predefined colors in Stat 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	

Using Nominal Variable Values to Color Markers

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*) 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 );
```

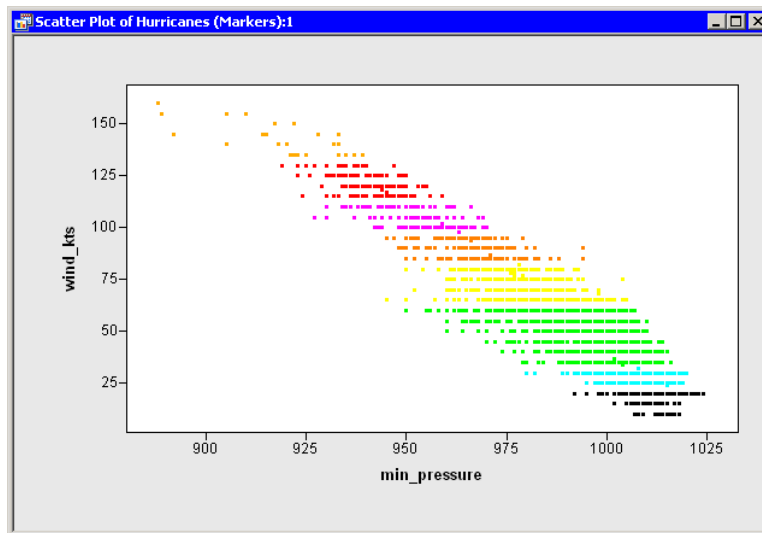


Figure 10.2. Markers Colored by a Nominal Variable

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

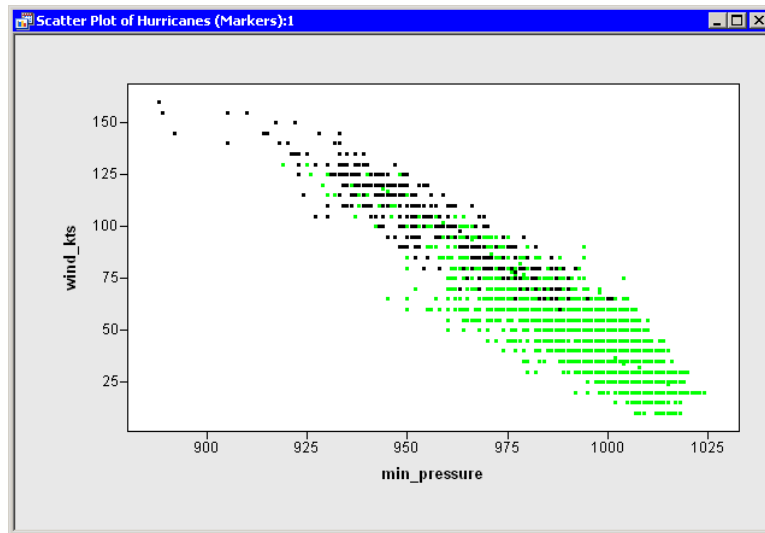


Figure 10.3. Markers Colored Manually

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 an IML matrix called `rEye`. The IML `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.

Changing 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, the `DataObject`'s `SetMarkerShape` method enables you to change a marker's shape. 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 IML `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 an IML matrix called `rEye`. The IML `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 "×." 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.

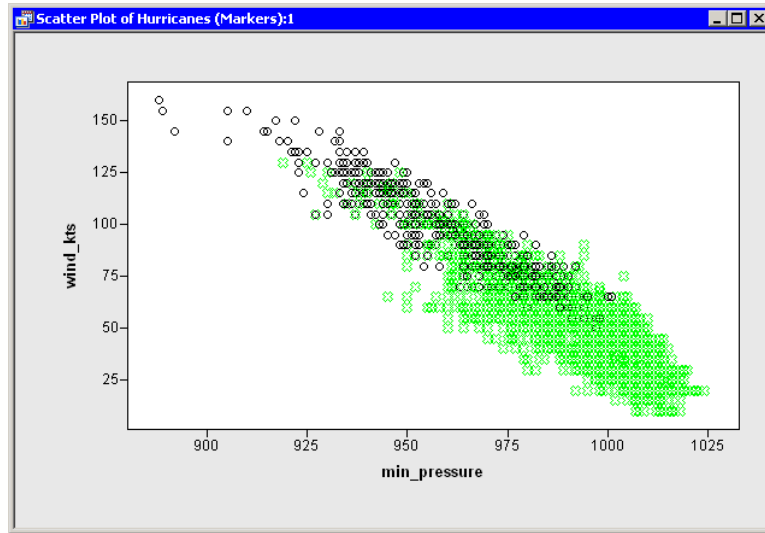


Figure 10.4. Changing Marker Shapes

In this example you used the `MARKER_X` and `MARKER_CIRCLE` shapes. The complete list of valid Stat 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>	★

Showing 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.

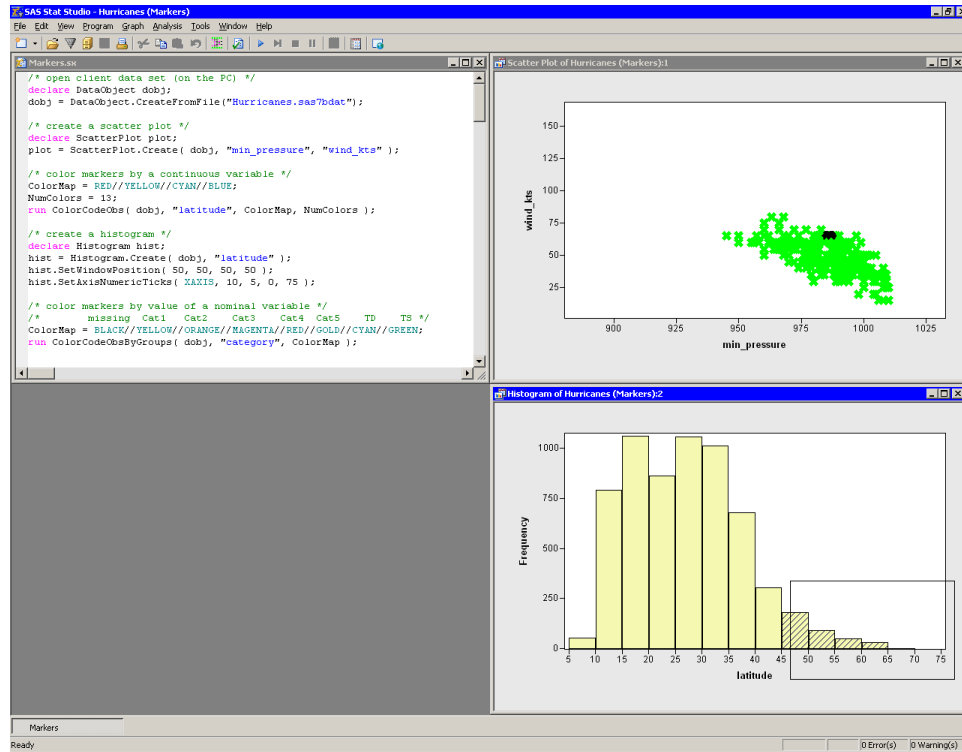


Figure 10.5. Showing Only Selected Observations

Index

A

- adding variables, [17](#)
- AddVar method, [17](#)
- AddVar.sx, [29](#)
- AddVars method, [18](#)
- APPEND statement, [17](#)
- AutoExec.sas, [11](#)
- Axes.sx, [47](#)
- axis view range, [49](#), [53](#)

B

- base class, [22](#), [32](#)

C

- case-sensitive, [6](#), [16](#)
- class, [5](#)
- client, [8](#), [9](#)
- color map, [57](#)
- ColorCodeObs module, [55](#)
- ColorCodeObsByGroups module, [57](#)
- CONCAT function, [43](#)
- continuous variables,
 - See interval variables
- coordinate system, [32](#)
- CopyServerDataToDataObject module, [15](#), [18](#), [30](#)
- Create method, [17](#)
- CREATE statement, [17](#)
- CreateFromFile method, [14](#)
- CreateFromServerDataSet method, [15](#)
- creating a DataObject
 - from client data sets, [10](#), [14](#)
 - from IML Matrices, [17](#)
 - from server data sets, [11](#), [15](#)

D

- Data views, [7](#)
- Data.sx, [14](#)
- DataObject class, [6](#)
 - purpose, [7](#)
- DataObject methods
 - AddVar, [17](#)
 - AddVars, [18](#)
 - Create, [17](#)
 - CreateFromFile, [5](#), [14](#)
 - CreateFromServerDataSet, [15](#)
 - GetVarData, [18](#), [33](#), [58](#)
 - SetMarkerColor, [58](#)
 - SetMarkerFillColor, [60](#)
 - SetMarkerShape, [60](#)

- Sort, [5](#)
- WriteToFile, [15](#)
- WriteToServerDataSet, [16](#)
- WriteVarsToServerDataSet, [16](#)
- DataTable class, [14](#)
- DataView class, [22](#)
- DataView methods
 - SetWindowPosition, [22](#), [52](#)
- DELETE subroutines, [19](#)
- discrete variables,
 - See nominal variables
- dot notation, [5](#)
- DrawInset module, [44](#)
- DrawLegend module, [43](#)
- DrawLine method, [32](#)
- DrawSetPenColor method, [34](#)
- DrawSetPenStyle method, [32](#)
- DrawUseDataCoordinates method, [32](#)
- dynamically linked, [7](#), [9](#), [21](#)

E

- ENDSUBMIT statement, [17](#), [26](#)

F

- file extensions, [14](#), [15](#)
- Fit.sx, [31](#)

G

- GetPersonalFilesDirectory module, [16](#)
- GetVarData method, [18](#), [33](#), [58](#)
- Graphs.sx, [21](#)

H

- help,
 - See online Help
- Help ► Help Topics, [3](#)
- histogram bin width, [52](#)
- Histogram class, [22](#)
- Histogram methods
 - Rebin, [54](#)
- Hurricanes data set, [10](#)

I

- IMLPlus, [2](#)
 - programs, [4](#)
- IMLPlus keywords
 - colors, [57](#)
 - declare, [5](#)
 - line styles, [32](#)

marker shapes, 61
 in-memory data, 5, 6
 insets, 41, 44
 installation directory, 10
 interval variables, 55

K

KDE procedure, 36

L

legends, 41, 43
 LIBNAME statement, 12, 15
 library, 13
 librefs, 11
 defining, 15
 line styles, 32
 LOC function, 58, 60

M

markers
 color, 55, 57
 hollow, 60
 predefined shape, 61
 shape, 60
 size, 60
 Markers.sx, 55
 methods, 5
 minor tick marks, 48
 missing values
 color, 57
 not colored, 56

N

nominal variables, 57
 nonprinting columns, 39

O

object-oriented, 5
 objects, 5
 OBS_ALL, 59
 ODS statement, 37
 ODS table name, 37
 ODS.sx, 37
 online Help, 3, 6, 14, 15, 23, 32, 43, 44, 48, 57
 opening
 client data sets, 10, 14
 server data sets, 11, 15
 output data sets, 30
 Output Delivery System(ODS), 37
 output windows, 37

P

personal files directory, 10, 11, 16
 Plot methods
 DrawLine, 32
 DrawSetPenColor method, 34
 DrawSetPenStyle method, 32
 DrawUseDataCoordinates, 32

SetAxisLabel, 48
 SetAxisMinorTicks, 48
 SetAxisNumericTicks, 54
 SetAxisTickAnchor, 48
 SetAxisTickRange, 54
 SetAxisTicks, 50
 SetAxisTickUnit, 48
 SetMarkerSize, 60
 SetTitleText, 43
 ShowAxisReferenceLines, 50
 ShowTitle, 43

polylines, 33
 predefined colors, 57
 Proc.sx, 25
 program windows, 4, 14
 programming language, 2
 programs, 3

AddVar.sx, 29
 Axes.sx, 47
 Data.sx, 14
 Fit.sx, 31
 Graphs.sx, 21
 Markers.sx, 55
 ODS.sx, 37
 Proc.sx, 25
 Titles.sx, 41

PUTN function, 43

Q

querying data, 6

R

READ statement, 17, 35, 40
 ReBin method, 54
 REG procedure, 26, 30, 31

S

Saffir-Simpson intensity scale, 50
 SAS library, 9
 saving
 data to a SAS library, 13, 16
 data to the client, 12, 15
 ScatterPlot class, 21
 selecting observations, 22
 selection rectangle, 22
 server, 8, 9
 name, 11
 SetAxisLabel method, 48
 SetAxisMinorTicks method, 48
 SetAxisNumericTicks method, 54
 SetAxisTickAnchor method, 48
 SetAxisTickRange method, 54
 SetAxisTicks method, 50
 SetAxisTickUnit method, 48
 SetMarkerColor method, 58
 SetMarkerFillColor method, 60
 SetMarkerShape method, 60
 SetMarkerSize method, 60
 SetTitleText method, 43

- SetWindowPosition method, [22](#), [52](#)
- show only selected observations, [61](#)
- ShowAxisReferenceLines method, [50](#)
- ShowTitle method, [43](#)
- SORT call, [33](#)
- SORT procedure, [36](#)
- SUBMIT statement, [17](#), [26](#)

T

- titles, [41](#), [43](#)
- Titles.sx, [41](#)

U

- unicode characters, [v](#)
- USE statement, [17](#), [35](#), [40](#)

W

- WriteToFile method, [15](#)
- WriteToServerDataSet method, [16](#)
- WriteVarsToServerDataSet method, [16](#)

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **`yourturn@sas.com`**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **`suggest@sas.com`**.

SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at support.sas.com/bookstore.

SAS® Press

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – **free** on the Web.
- Hard-copy books.

support.sas.com/publishing

SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

support.sas.com/spn



**THE
POWER
TO KNOW®**

