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

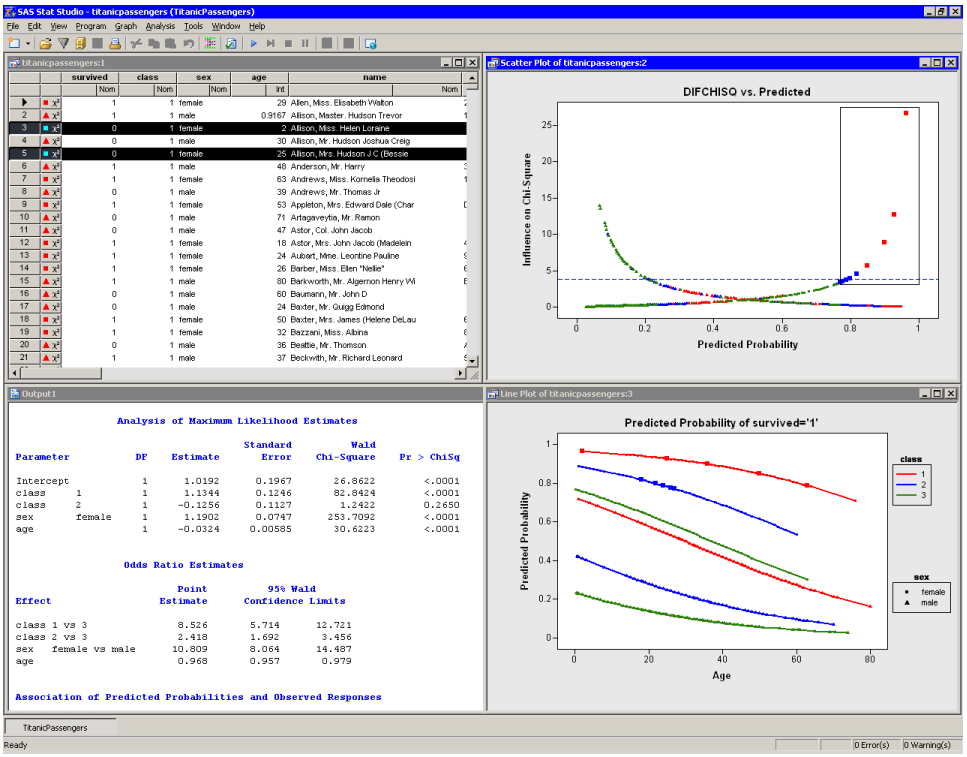

Figure 1.1. The Stat Studio Interface

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.

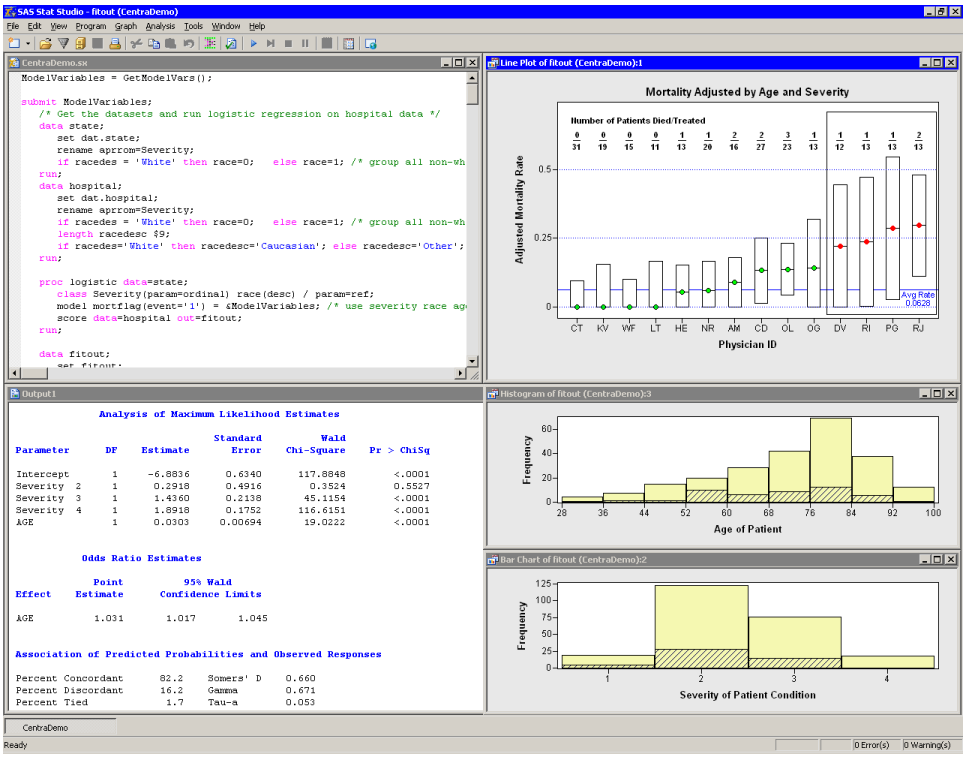

Figure 1.2. Results of an IMLPlus Program

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 provide 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.

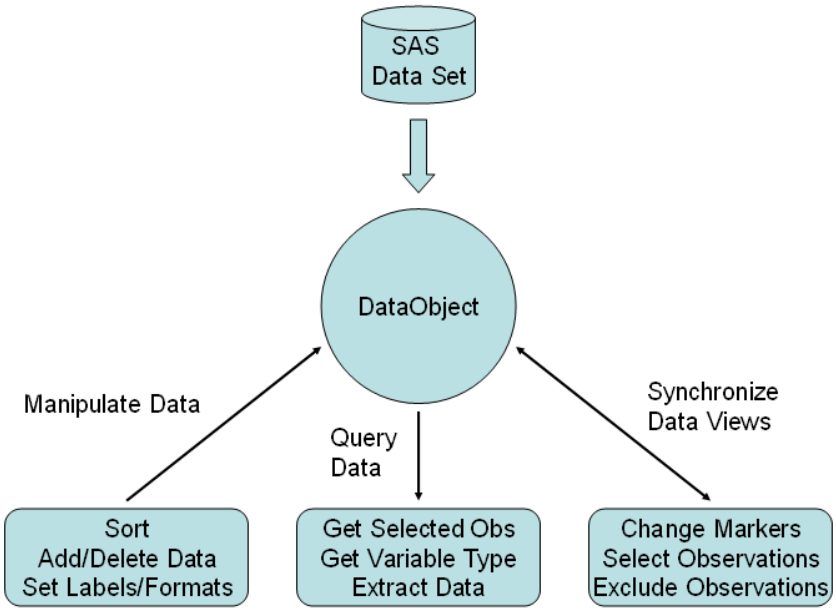

Figure 1.3. Using a DataObject

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.

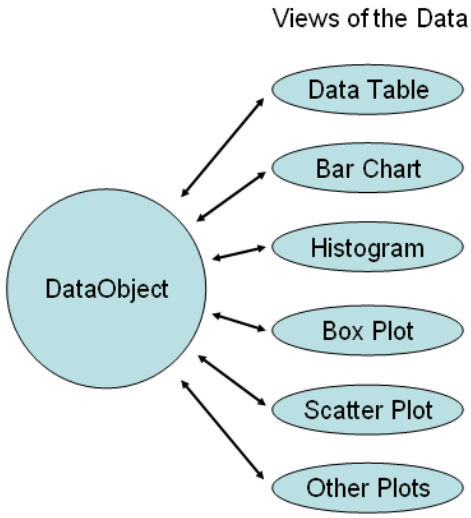

Figure 1.4. The DataObject Role

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.

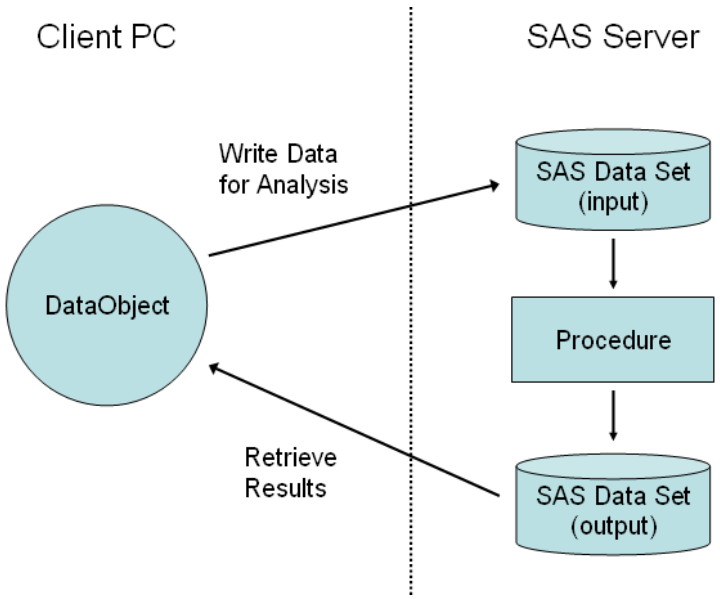

Figure 1.5. Data Flow

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.