CHAPTER 1

Introduction to Object-Oriented Statistical Programming

Introduction

An object-oriented approach to statistical programming and analysis focuses attention on the content of the statistical model, but not on the details of the computation. This programming and analysis style enables you to perform analyses using only statistical categories, while making computations transparent. Such a style is implemented through the use of classes and the relationships between them.

Here are definitions for the main terms that we use in the book:

class	the template or model for a statistical object, which includes data describing the object's characteristics (attributes) and actions that it can perform.
attribute	a characteristic associated with a statistical object. All objects of the same class have the same set of attributes. These attributes are specified by name, type, and initial value, and they are automatically initialized when an object is created.
action	an operation that is defined for a class and can be executed by any object created from that class.

A specific representation of a class is called an object, and a specific implementation of an action for a specific class is called a method. An object inherits all the attributes of its class as well as the actions that the class can execute.

In this book we consider only three kinds of relationships between classes: inheritance, superclass/subclass, and abstract superclass. Inheritance is the mechanism that allows class *One* to inherit the attributes and actions of class *Two* without the need to recreate/redefine them. The class *One* should supply only attributes and actions that need to be different from those inherited. If class *One* inherits from class *Two*, then *Two* is called superclass of *One*, and *One* is called subclass of *Two*. With inheritance, we enable a subclass to offer the same properties as their superclasses. Class *Two* can be an abstract superclass if it is only used as a superclass for other classes. In this case, class *Two* only specifies attributes, and its subclasses must define the attributes of *Two*.

To highlight the differences between a class and an object, and between an action and a method, we use special style conventions (see "Conventions" in the Preface). For example: "The **vector** object of the *Vector Discrete* class uses the _dcreate() method that implements the Create action.

We have implemented the object-oriented approach to statistical programming and analysis using the previously developed table-driven environment described in our earlier book[1]. The table-driven environment provides you with a convenient and reliable environment for statistical object-oriented programming and analysis. The table-driven environment contains the data dictionary and its supporting programs. At the heart of this environment is the set of specially structured tables that form the data dictionary. The data dictionary contains information concerning classes, their attributes, and actions (Chapter 4 describes the data dictionary in detail). The SAS System with our table-driven environment allows you to define classes of objects. Using this environment, you can easily and quickly extend the set of already-created classes.

In this chapter, we guide you through several simple examples that show how to implement a mechanism of object-oriented statistical programming and analysis with the SAS System. The use of the actions allows programming in SAS to have much more of the style of object-oriented programming systems. However, because we use the table-driven environment, methods that implement actions are not restricted to automatic invocation, and they can be used like any other SAS macro function.

Fundamentals of Object-Oriented Programming

In object-oriented programming, you consider a class of objects and try to imagine all the actions you may want to perform on such objects. You then define data attributes and actions specifically for that class of objects.

In this book, the type of object will be specified only when it is necessary to distinguish between them, or when the type of object is not clear from the context. Otherwise, 'object' refers to both data and statistical objects.

For example, suppose you want to create an array of numbers as a statistical object, and you want to print the data contained in that object. Using the object-oriented approach to statistical analysis, you can define a class of objects called *Vector*, and then define actions for generating an object of this class and printing data that is contained by this object. These two actions, just as an example, can be defined for a wide variety of classes, but might need to be implemented differently for each of them.

An action that is common to a wide variety of classes is called a generic action. The actual implementation of an action for a specific class is called a method. Actions are carried out by means of a mechanism that identifies the class of arguments and calls the appropriate methods.

The definition of a new class is performed in the tables of the data dictionary. The data dictionary is a set of tables with strictly defined relations that stores definitions of classes, methods, objects, and so on. (Chapter 4 describes the data dictionary in detail.) Each class, its attributes, and its methods should be defined in three tables: Class, ClassAtr, and ClassMet. In Chapter 2 we will discuss in detail how to define a new class

[1] Kolosava, Tanya and Berestizhevsky, Samuel, *Table-Driven Strategies for Rapid SAS Applications Development*, Cary, NC: SAS Institute, Inc., 1995

in the data dictionary, but here is a very simple example of a definition of the *Vector* and *Vector Discrete* classes. The Class table defines and describes the classes:

Class Table

CLASS	SPRCLASS	Description
Vector	Main	Array of continuous numbers
Vector Discrete	Vector	Array of discrete numbers

These two rows in the Class table define the following:

- There is a *Vector* class that specifies attributes and actions on an array of continuous numbers. This class is a subclass of the *Main* class (an abstract superclass that will be defined in Chapter 2).
- There is a *Vector Discrete* class that specifies attributes and actions on an array of discrete numbers, and it is derived from the *Vector* class.

The ClassAtr table contains a definition of class attributes for each class:

ClassAtr table

CLASS	ATTR_NO	ATTRNAME	ATTRDESC
Vector	1	OBJECT	Name of the data object producing an object of the Vector class.
Vector	2	COLUMN	Name of the column of the data object.

Note that we have to define attributes only for the *Vector* class – the *Vector Discrete* class inherits these attributes.

Finally, the ClassMet table stores the definitions of classes, their actions, and the methods for implementing actions:

ClassMet table

CLASS	ACTION	METHOD	ACTDESC
Vector	CREATE	_vcreate()	Creates an object of the Vector class
Vector	PRINT	_vprint()	Prints data of an object of the Vector class
Vector Discrete	CREATE	_dcreate()	Creates an object of the Vector Discrete class
Vector Discrete	PRINT	_dprint()	Prints data of an object of the Vector Discrete class

In this simple example we define only two actions for the *Vector* and *Vector Discrete* classes: Create and Print actions. Implementation of the actions is different for these classes despite the subclass/superclass relationship between the vector and vector discrete classes.

Because the object-oriented programming is implemented with the SAS System, we store the tables that form the data dictionary (for example, the Class, ClassMet, and ClassAtr tables) as SAS data sets, and we implement methods as SAS macro programs.

Let's create a new **myvector** object of the *Vector* class. To do this, the **myvector** object should be associated with the *Vector* class in the tables of the data dictionary. This data dictionary has two tables that define statistical objects and all actions that can be performed on these objects.

The StatObj table stores all the information that is necessary for the definition of a new object. The columns of the StatObj table are defined as follows:

Columns of the StatObj table

Column name	Type	Length	Description
STATOBJ	Character	8	Name of the statistical object
CLASS	Character	8	Name of the class
ATTR_NO	Numeric	8	Order number of the class attribute
ATTR_VAL	Character	20	Value of the class attribute

The STATOBJ, CLASS and ATTR_NO columns form the primary key of the StatObj table.

Each new object should be defined in the StatObj table. The definition of the **myvector** object of the *Vector* class in the StatObj table looks like this:

StatObj table

STATOBJ	CLASS	ATTR_NO	ATTR_VAL
myvector	vector	1	cars
myvector	vector	2	mileage

This table defines a new object of *Vector* class. The name of the object is **myvector**. Recall that, for the purposes of this example, we have mentioned only two parameters: the name of the data object (the first value of the ATTR_VAL column) and the name of the column (the second value of the ATTR_VAL column) that produce the **myvector** object. As you can see for this definition, the new statistical object should be produced from the MILEAGE column of the Cars table. The Cars table can be a SAS data set, or it can be any type of data set that the SAS System can access (for example, a table created by a relational database such as Rdb, SYBASE, ORACLE, a spreadsheet table; or an external file).

The StatAct table contains the definition of actions that should be performed on a statistical object. The columns of the StatAct table are defined as follows:

Columns of the StatAct table

Column name	Type	Length	Description
ANALYSID	Character	8	Identification name of designed statistical analysis
ORDER	Numeric	8	Order number
STATOBJ	Character	8	Name of the statistical object.
ACTION	Character	20	Name of the action to be performed
PARAMS	Character	80	Comma-delimited list of parameters

The ANALYSID and ORDER columns form the primary key of the StatAct table.

In order to define actions for the new **myvector** object, you should fill in the StatAct table as follows:

StatAct table

ANALYSID	ORDER	STATOBJ	ACTION	PARAMS
example1	1	myvector	CREATE	myvector
example1	2	myvector	PRINT	

The definitions stored in the StatAct table mean that Create and Print are the actions that you can perform on the **myvector** object.

Actually, each action is performed by a single macro program %ACTION. You will find a detailed description and the code for this program in Chapter 4. Here we just mention that the function of the %ACTION macro program is to find an appropriate method and to apply it to the object.

The %ACTION macro receives three named parameters: `object`, `action`, and `params`. The `object` parameter gets the name of the statistical object to be processed. The `action` parameter gets the name of the action to be applied to the object. The `params` parameter gets a quoted string containing parameters of the action, if any. For example, the creation of the **myvector** object is performed by submitting the following macro:

```
%action(object=myvector, action="Create",
params="myvector");
```

This macro would automatically submit the _vcreate() method (see the ClassMet table) implementing the Create action specifically for the *Vector* class.

Now we will look in greater detail at two aspects of the Create action:

1. Creation of the object is a generic action that exists for each class.

2. Creation of an object requires additional definitions in the Object and Property tables of the data dictionary.

Each time that you want to create a new object, you have to define, in terms of data objects, where and how the data of this object is to be stored. You make this definition in the Object and Property tables.

The Object table lists data objects and the names of corresponding SAS data sets where their data is stored. The Property table specifies the properties of data objects, listing the columns and characteristics (name, type, length, and so on) of each data object.

In our example, we specified that the **myvector** object of the *Vector* class should be produced from the MILEAGE column of the Cars table (see the StatObj table earlier in this chapter). Thus, the Cars table and at least one of its columns, MILEAGE, should be defined in the Object and Property tables, like this:

Object table

OBJECT	DATASET	TITLE	LIBRARY
cars	cars	Automobile Data Table	automob

Property table (selected columns)

OBJECT	COLUMN	TITLE	TYPE	LENGTH	ATTRIBUT
…	…	…	…	…	…
cars	MILEAGE	Mileage data	N	8	

This definition describes the physical location of the Cars table, and specifies its column (only one in this example). The _vcreate() method of the *Vector* class will take this information and check whether the CARS SAS data set that corresponds to the Cars table exists. If not, an empty data set that corresponds to the definitions in the Object and Property tables will be created. The %_VCREATE macro is available from SAS Institute, through the SAS Online Samples facility (for detailed information, refer to the inside back cover of the book).

Suppose that you need to create a new object of the *Vector Discrete* class, say **newvector**. Again, assume that the *Vector Discrete* class is already defined.

First, we'll define this new object in the StatObj table (see the highlighted rows):

StatObj table

STATOBJ	CLASS	ATTR_NO	ATTR_VAL
myvector	vector	1	cars
myvector	vector	2	mileage
newvector	vector discrete	1	cars
newvector	vector discrete	2	weight

The highlighted rows define a new object of the *Vector Discrete* class. The name of the object is **newvector**. For simplicity, let's consider only two parameters of the *Vector Discrete* class: the name of the table and the name of the column that produces the **newvector** object. According to this definition, the new statistical object should be produced from the WEIGHT column of the Cars table.

In order to define actions for the **newvector** object, you should add to the StatAct table the following definition (highlighted rows):

StatAct table

ANALYSID	ORDER	STATOBJ	ACTION	PARAMS
example	1	myvector	Create	myvector
example	2	myvector	Print	
example	3	newvector	Create	newvector

The definition that is stored in the third row of the StatAct table means that the new object **newvector** should be created by the Create action.

As it is defined in the StatObj table, the **newvector** statistical object should be produced from the WEIGHT column of the Cars table. Thus, you should update the Property table like this (highlighted row):

Property table (selected columns)

OBJECT	COLUMN	TITLE	TYPE	LENGTH	ATTRIBUT
…	…	…	…	…	...
cars	MILEAGE	Mileage data	N	8	
cars	WEIGHT	Weight of a car	N	8	

After you have finished all the definitions, you should submit the following macro:

```
%action(object=newvector, action = "Create",
params="newvector");
```

to create the **newvector** object of the *Vector Discrete* class.

As you can see, an object-oriented approach that is implemented through a table-driven environment provides a mechanism that allows one macro program, for example %ACTION, to be re-used to process many different classes of objects. Such a macro is called a *generic* macro, and it calls appropriate methods, in turn, for each object that it processes. For example:

```
%action(object=newvector, action="Create",
params="newvector") ;
```

is a matching method for object **newvector**, class *Vector Discrete,* and macro _dcreate(). The object itself contains information about which class it corresponds to.

Consider another example, a statistical class: *Hypothesis Testing*. This is a class of objects that results from performing a test that relates to statistical inference. An object of the *Hypothesis Testing* class represents a test through the estimation of a test statistic given null and alternative hypotheses and a confidence level. The methods that are used in estimating these test statistics vary from classical to robust and nonparametric approaches. You want to be able to create objects of the *Hypothesis Testing* class. Such an object may be implemented as a list (see Chapter 2) with the following attributes:

Attribute	**Description**
statistic	the value of the test statistic, with a NAMES attribute indicating its null distribution.
parameters	the parameters of the null distribution of the statistic.
p-value	the p-value for the test under the null hypothesis.
estimate	an estimate of the corresponding population parameters about which you formulated a null hypothesis. The attributes estimate has a names attribute describing its elements.
null-value	the value of the population parameter that is specified by the null hypothesis. The attribute component null-value has a names attribute describing its elements.
alternative	the value of the input argument alternative: "greater", "less", or "two-sided".
method	the name of the test used.
data name	the name of the input data object.

Introduction to Object-Oriented Statistical Analysis

In statistical analysis it's quite natural to operate by classes, because statistical methods are always linked to data. The advantages of object-oriented statistical programming and analysis are not evident when you are doing a very simple statistical analysis. The advantages arise when you are designing a complex statistical analysis that will perform similar, but not identical, operations on a variety of data objects. By specifying classes of data objects for which identical effects will occur, you can define a single generic macro function that embraces the similarities across object classes, but permits individual implementations (methods) for each defined class.

For example, PROC MEANS produces simple univariate descriptive statistics for numeric variables, and if you type the following:

```
proc means data = v_examp mean ;
var var1 ;
run ;
```

you expect SAS to calculate the mean of the var1 variable according to the discrete numeric type of this variable. However, suppose that you also want to calculate the mean of the var1 variable as an integer value. In order to get what you want, you have to write a simple DATA step as follows:

```
data _null_ ;
      retain mean 0 ;
      set v_examp nobs = last;
      mean = mean + var1 ;
      if _n_ = last then do ;
            mean = int(mean/last + 0.5) ;
            put "Mean of Var1 = " mean ;
      end ;
run ;
```

Such a DATA step must be modified every time a new class of objects is created. Using an object-oriented approach, you can use truly generic methods; they do not have to be modified to accommodate new classes of objects. The objects carry their own methods with them. Thus, when you create a class of objects, you can also create a set of methods to specify how these objects will behave with respect to a certain generic statistical operation.

As an example, let's consider the way that SAS will compute means for continuous and discrete numeric variables using the generic %ACTION macro. If you call the following macro:

```
%action(object=newvector, action="mean");
```

then the %ACTION macro computes the mean of the **newvector** object as a discrete value, because the **newvector** object is of the *Vector Discrete* class. By contrast, in this case:

```
%action(object=myvector,  action="mean");
```

the same macro computes the mean as a continuous value, because the **myvector** object belongs to the *Vector* class.

The %ACTION generic macro program evaluates tables of the data dictionary such as ClassMet, ClassAtr, StatObj, StatAct, Object, and Property; identifies the class of the object and its attributes; finds the appropriate method; and invokes the macro program implementing this method that, finally, processes data stored in the SAS data set.

Because data exists in the statistical object as a reference to an application table, it is possible to use the same application table for different objects. Even the same column of an application table can be referred to in different statistical objects.

For example, if you want to consider the WEIGHT column of the Cars table (shown below) once as a discrete vector and once as a continuous numeric vector, you should define two statistical objects referring to the same data object. Look at the definition of the **r_vector** object in the StatObj table (highlighted rows):

StatObj table

STATOBJ	CLASS	ATTR_NO	ATTR_VAL
myvector	vector	1	cars
myvector	vector	2	mileage
newvector	vector discrete	1	cars
newvector	vector discrete	2	weight
r_vector	vector	1	cars
r_vector	vector	2	weight

The **newvector** and **r_vector** objects that are defined in the StatObj table will be processed differently, although they refer to the same data object. Thus, if you define desired actions on **newvector** and **r_vector** objects like this :

StatAct table

ANALYSID	ORDER	STATOBJ	ACTION	PARAMS
example1	3	newvector	Create	newvector
example1	4	newvector	Mean	
example1	5	r_vector	Create	r_vector
example1	6	r_vector	Mean	

the results of the “Mean” action may be different for the **newvector** and **r_vector** objects, although the action was performed on the same column (WEIGHT) of the same table (Cars).

Developing Statistical Models

All statistical analysis models attempt to describe the structure or relationships of some objects from which data is derived. Modern statistical analysis provides an extremely rich choice of modeling techniques. The choice of a modeling technique depends on the type and structure of your data and what you want the model to test or explain. The development of statistical models is data dependent. The process of developing a statistical model also depends on whether you follow a hypothesis-driven approach (confirmatory data analysis) or data-driven approach (exploratory data analysis). Data analysts frequently combine both approaches.

For example, in classical hypothesis-driven regression analysis, you usually examine residuals using exploratory data analysis methods. The goal of each approach is a model that imitates, as closely as possible, the properties of the real objects being modeled. The differences between model and reality, the residuals, are often the key to reaching a deeper understanding and obtaining a better model. Creating a statistical model usually involves the steps shown in Figure 1:

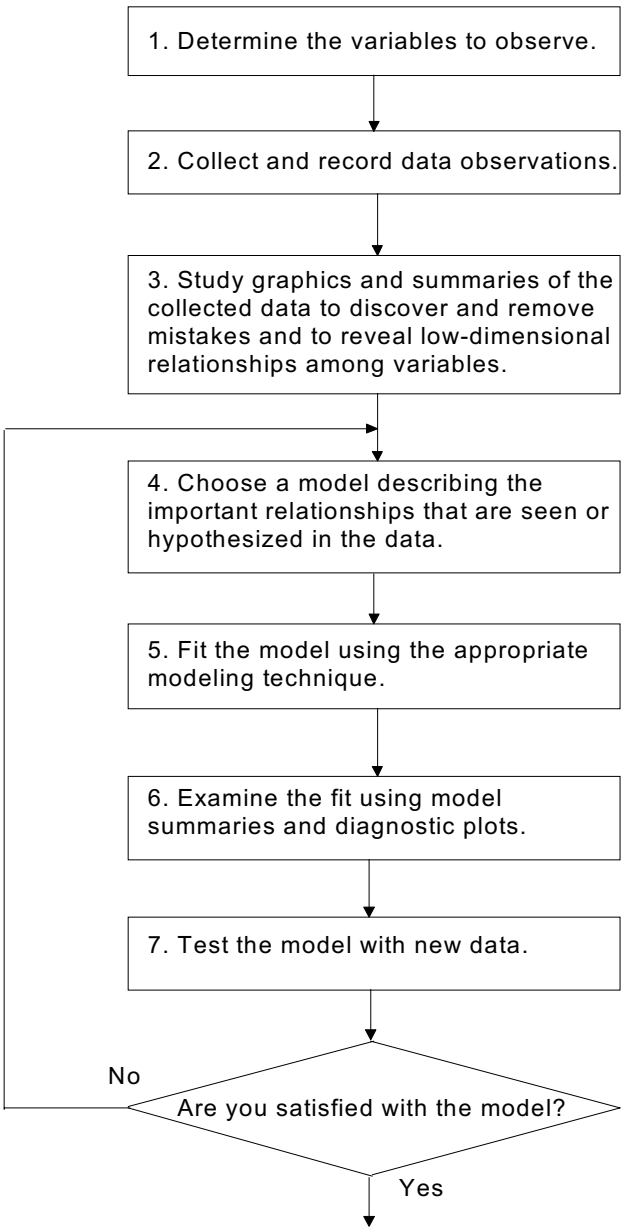

Figure 1.1 Flow chart of the modeling process

At any point in the modeling process described in Figure 1, you may find that your choice of a model does not appropriately fit the data.

In some cases, diagnostic plots (step 6) may give you clues to improve the fit. Sometimes you may need to try transformed variables (step 3) or entirely different variables (step 1). You may need to try different modeling techniques that will allow you to fit in nonlinear relationships or interactions (step 5). At times, all you need is to remove outlying, influential data (step 3), or fit the model robustly (step 5).

Of course, there is no one answer for how to build good statistical models. By iteratively fitting, plotting, testing, changing something, and then refitting, you will arrive at the best fitting model for your data.

When developing a statistical model in the SAS System, there are a wide range of possible modeling techniques to choose from. Among them are generalized linear models, analysis of variance, autoregressive models, and many more. Fortunately, many different classes of statistical models share a substantial common structure. The steps listed in Figure 1.1 apply to many models, and important summaries and diagnostics can be shared directly or adapted straightforwardly from one class of models to another. The object-oriented statistical programming and analyses of the various classes of models take advantage of this common structure.

Data Used for Models

Statistical models allow inferences to be made about objects by modeling associated data, organized by variables. A SAS data set can be considered an object that represents a sequence of observations on some chosen set of variables. SAS data sets allow computations where variables can act as separate objects and can be referenced simply by naming them. This make SAS data sets very useful in modeling.

Variables in SAS data sets are of two types: Numeric and Character. These two types do not cover all data types that you need in statistical modeling. However, by associating a SAS data set variable with some class, you specify how this variable should be considered. For example, associating a numeric variable with the *Vector Discrete* class defines values of this variable as discrete. At the same time, if you associate the same numeric variable with the *Vector* class, the values of this variable will be interpreted as continuous.

When developing a model, the types of data you have are important for deciding which modeling technique best suits your data. Continuous data represents quantitative data having a continuous range of values. Categorical data represents qualitative data (discrete data), meaning that they can assume only certain fixed numeric or character values. In object-oriented statistical analysis you represent categorical data with a factor, which keeps track of the levels or different values that are contained in the data and the level each data point corresponds to. Numeric objects in object-oriented statistical analysis are vectors, or matrices.

A statistical model expresses a response variable as some function of a set of one or more predictor variables. The type of model you select depends on whether the response and predictor variables are continuous or categorical. For example, the classical regression problem has a continuous response and continuous predictors, but the classical analysis-of-variance problem has a continuous response and categorical predictors.

A data property that is very important for statistical modeling and inference is the level of data measurement, or the measurement scale. There are four different scales:

1. Nominal scale: classifies each observation according to specific characteristics. For example, sex (male/female), political party (Democrat/Republican/Independent/Other), binary measures (success/failure).

2. Ordinal scale: classifies each observation according to specific characteristics, but with some ordering. For example, number of cigarettes smoked (0, <1 pack/day, 1–2 packs/day, > 2 packs/day), feelings on an issue (disagree, indifferent, agree), military rank (Captain, Major, Colonel, General).

3. Interval scale: ordering is inherent in the data, and even more importantly, there is a common and constant unit that is used for the measurement. For example, temperature scales such as Celsius or Fahrenheit (but not Kelvin). Zero is not important.

4. Ratio scale: ratios are meaningful, for example 50K is half of 100K. Additional examples include temperature on the Kelvin scale, monetary units, weight, distance, and velocity.

Example of Data Analysis

The example that follows describes only one way of analyzing data through the use of statistical modeling. There is no perfect cookbook approach to building models, as different techniques do different things, and not all of them use the same arguments when doing the actual fitting. The following analysis uses a data set that contains a variety of data for car models. The complete data set is available through the SAS Online Samples facility (for detailed information, refer to the inside back cover of the book).

This is how the data set looks:

Cars table

COUNTRY	TYPE	RELIABIL	MILEAGE	WEIGHT
Japan	Small	5	.	2700
Japan	Medium	5	20	3265
Germany	Medium	.	.	2935
Germany	Compact	.	27	2670
Germany	Compact	4	.	2895
Germany	Medium	.	.	3640
USA	Medium	3	21	2880
USA	Large	3	.	3350
USA	Large	3	23	3325
…	…	…	…	…

Next we show you how to define the statistical analysis of this data using an object-oriented approach. First, let's update information about the **cars** data object in the Property table. The updated table looks like this:

Property table (selected columns)

OBJECT	COLUMN	TITLE	TYPE	LENGTH	ATTRIBUT
…	…	…	…	…	…
cars	COUNTRY	Country	C	20	P
cars	TYPE	Type of a car	C	10	P
cars	RELIABIL	Reliability mark	N	8	
cars	MILEAGE	Mileage data	N	8	
cars	WEIGHT	Weight of a car	N	8	

The value "P" for the ATTRIBUT column of the Property table defines the COUNTRY and TYPE columns as primary key of the Cars table.

Now we can define statistical objects in the StatObj table. Once more, let's assume for simplicity that all statistical classes that we will use in this example are already defined in the tables of the data dictionary, and each of them has only two parameters: data object name and column name. The definitions of the new statistical objects in the StatObj table look like this:

StatObj table

STATOBJ	CLASS	ATTR_NO	ATTR_VAL
mileage	vector	1	cars
mileage	vector	2	mileage
reliabl	vector discrete	1	cars
reliabl	vector discrete	2	mileage
weight	vector	1	cars
weight	vector	2	weight
country	factor	1	cars
country	factor	2	country
type	factor	1	cars
type	factor	2	type

As you see, the definitions of the statistical objects are very obvious and need no additional comments.

Now we are ready to define statistical processing of the new objects. To do it, we just need to define in the StatAct table which actions we want to be done:

StatAct table

ANALYSID	ORDER	STATOBJ	ACTION	PARAMS
example2	1	mileage	Create	mileage
example2	2	reliabl	Create	reliabl
example2	3	weight	Create	weight
example2	4	type	Create	type
example2	5	country	Create	country
example2	6	mileage	Min	
example2	7	mileage	Mean	
example2	8	mileage	Median	
example2	9	mileage	Max	
example2	10	reliable	Mean	
example2	11	reliable	Median	
example2	12	weight	Min	
example2	13	weight	Mean	
example2	14	weight	Median	
example2	15	weight	Max	
example2	16	country	Group	
example2	17	type	Subgroup	

Now the actions will be performed according to the definitions in the StatAct table. You can perform the statistical analysis in the following manner:

1. Create the new statistical objects by using the %ACTION macro as follows:

```
%action(object=mileage, action = “Create”,
params=“mileage”);

%action(object=reliabl, action = “Create”,
params=“reliabl”);

%action(object=weight, action = “Create”,
params=“weight”);

%action(object=type, action = “Create”,
params=“type”);

%action(object=country, action = “Create”,
params=“country”);
```

2. Calculate descriptive statistics by using the %ACTION macro as follows:

```
%action(object=reliabl, action = “Median”);

%action(object=weight, action = “Mean”);
```

The desired descriptive statistics will be calculated – each time by an appropriate method.

However, you should not call the %ACTION macro program for each action. This job will be done for you by the macro program %ANALYZE, which gets the ANALYSID value as its parameter. The %ANALYZE macro is available through the SAS Online Samples facility (for detailed information, refer to the inside back cover of the book).

Summary

In this chapter we recalled the fundamentals of an object-oriented approach and discussed statistical modeling in general terms. The main goal was to demonstrate how to apply this approach to statistical programming and analysis, and how it can be implemented in SAS using a table-driven environment. Briefly, here are the main ideas:

- Specially-structured tables form a data dictionary that enables you to define statistical classes, to define instances of these classes (that is, statistical objects), and to define statistical processing of these instances.

- A mechanism implemented with SAS macro language resolves the definitions stored in the data dictionary.

In the following chapters you will get a detailed explanation of each component in this approach. You will learn how to define statistical classes (Chapter 2) and how to create a set of actions that specify how a statistical object will behave with respect to certain generic operations (Chapter 3). You will learn the structure of the data dictionary and how to define classes and objects in the data dictionary tables (Chapter 4). And finally, you will solve some statistical problems using all you have learned (Chapter 5).