Programming Techniques for Data Mining with SAS

Samuel Berestizhevsky, YieldWise Canada Inc, Canada
Tanya Kolosova, YieldWise Canada Inc, Canada

ABSTRACT

Object-oriented statistical programming is a style of data analysis and data mining, which models the relationships among the objects that comprise a problem rather than procedures that can be taken to solve the problem. One of the key steps in moving to object-oriented statistical programming and analysis is to make a shift in thinking away from the procedural model to a new model where procedure execution is a result of relationships among objects. This new model is more declarative, meaning that objects are first declared then allowed to act or interact. This manner of actions makes it easier to design and implement statistical analyses and data mining which take the actions they do only as a reaction to object state.

In data analysis and data mining 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 themselves when you are doing a very simple statistical analysis. The advantages arise when you are designing a complex statistical analysis that will do similar, but not identical, things to 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.

The goals of this article are:
- to provide an understanding of concepts of object-oriented statistical programming and analysis
- to demonstrate SAS® software’s ability to implement object-oriented approach to data analysis and data mining

INTRODUCTION

The object-oriented approach to statistical analysis and data mining enables the data analyst/data miner, statistician and programmer to operate as an effective team for solving data analysis problems. The process of problem solving becomes much more structured and documented.

First, the data analyst/data miner structures data and defines a data object in the tables of specially structured data dictionary. Then this data object is associated with an appropriate statistical class. The data analyst/data miner uses actions of this statistical class to perform data analysis.

Such statistical classes are designed by the statistician. Understanding of a specific problem and statistical knowledge enable the statistician to generalize a problem and create a class that is suitable for a specific group of problems. Each time a new problem arises, that does not yet have an appropriate class, the statistician creates the new suitable class.

Once the programs supporting the object-oriented approach are developed, the programmer has only to implement methods for new classes, often re-using previously written code.

FUNDAMENTALS OF OBJECT-ORIENTED STATISTICAL PROGRAMMING

An object-oriented approach to statistical analysis and data mining means to focus 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 relationships between them.

Here are definitions of the main terms that we use in this article:

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

We will consider in this article two kinds of objects. The first is a data object that is just a data table with its properties -- name, corresponding SAS data set, columns and their characteristics. Second is a statistical object that can be defined as a data object with additional statistical characteristics. Usually, a statistical object contains a data object as one of its attributes. In the statements that are true for both data and statistical objects, we will say simply object. In the cases that the kind of an object is important and is not clear from the contents, we will specify it explicitly.

For example, consider an array of numbers (vector) as a statistical object. You want to be able to create this object, and to print data that it contains. 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 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 for 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 based on a mechanism of identifying the class of their arguments, and calling the appropriate methods. For example, for the Vector class you will define _vcreate() and _vprint() 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 relationships that store definitions of classes, methods, objects, etc. Each class, its attributes and methods should be defined in three tables: Class, ClassAtr and ClassMet. The Class table is intended to define a class, and to describe it:

<table>
<thead>
<tr>
<th>CLASS</th>
<th>SPRCLASS</th>
<th>Descrit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Main</td>
<td>Array of continous numbers.</td>
</tr>
<tr>
<td>Vector</td>
<td>Vector</td>
<td>Array of discrete numbers.</td>
</tr>
<tr>
<td>Discrete</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These two rows in the Class table define the following:

- There is Vector class that specifies attributes and actions on an array of continuous numbers. This class is a subclass of the Main class.
- There is Vector Discrete class that specifies attributes and actions on an array of discrete numbers and is derived from the Vector class.

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

<table>
<thead>
<tr>
<th>CLASS</th>
<th>ATTR_NO</th>
<th>ATTRNAME</th>
<th>ATTRDESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>1</td>
<td>object</td>
<td>Name of the data object producing an object of the Vector class</td>
</tr>
<tr>
<td>Vector</td>
<td>2</td>
<td>column</td>
<td>Name of the column of the data object</td>
</tr>
</tbody>
</table>

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

Finally, the ClassMet table stores definitions of classes, their actions and methods implementing actions:
For needs of this 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 of subclass/superclass relationships.

Because the object-oriented programming is implemented with the SAS® System, we store the tables forming the data dictionary (e.g. 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, besides others, has two tables that are intended to define statistical objects and all actions that can be performed on these objects.

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

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATOBJ</td>
<td>Character</td>
<td>8</td>
<td>Name of the statistical object</td>
</tr>
<tr>
<td>CLASS</td>
<td>Character</td>
<td>8</td>
<td>Name of the class</td>
</tr>
<tr>
<td>ATTR_NO</td>
<td>Numeric</td>
<td>8</td>
<td>Order number of the class attribute.</td>
</tr>
<tr>
<td>ATTR_VAL</td>
<td>Character</td>
<td>20</td>
<td>Value of the class attribute</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>STATOBJ</th>
<th>CLASS</th>
<th>ATTR_NO</th>
<th>ATTR_VAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>myvector</td>
<td>vector</td>
<td>1</td>
<td>cars</td>
</tr>
<tr>
<td>myvector</td>
<td>vector</td>
<td>2</td>
<td>mileage</td>
</tr>
</tbody>
</table>

This 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: name of the data object (the first value of the ATTR_VAL column) and name of the column (the second value of the ATTR_VAL column), that produce 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 can be any type of data set that the SAS® System can access (for example: a table of a relational database, 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

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANALYSID</td>
<td>Character</td>
<td>8</td>
<td>Identification name of designed statistical analysis</td>
</tr>
<tr>
<td>ORDER</td>
<td>Numeric</td>
<td>8</td>
<td>Order number</td>
</tr>
<tr>
<td>STATOBJ</td>
<td>Character</td>
<td>8</td>
<td>Name of the statistical object</td>
</tr>
<tr>
<td>ACTION</td>
<td>Character</td>
<td>20</td>
<td>Name of the action to be performed</td>
</tr>
<tr>
<td>PARAMS</td>
<td>Character</td>
<td>80</td>
<td>Comma-delimited list of parameters</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>analysid</th>
<th>order</th>
<th>STATOBJ</th>
<th>action</th>
<th>params</th>
</tr>
</thead>
<tbody>
<tr>
<td>example1</td>
<td>1</td>
<td>myvector</td>
<td>Create</td>
<td>myvector</td>
</tr>
<tr>
<td>example1</td>
<td>2</td>
<td>myvector</td>
<td>Print</td>
<td></td>
</tr>
</tbody>
</table>
```

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 (see Appendix for its source code). 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, submitting of the following macro performs the creation of the myvector object:

```
%ACTION(object=MYVECTOR, action="CREATE", params="MYVECTOR");
```

and would automatically submit the _vcreate() method (see the ClassMet table) implementing the action “Create” specifically for the Vector class.

Each time that you want to create a new object, you have to define, in terms of data objects, where and how data of this object are 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, it lists the columns and their characteristics (name, type, length, etc.) of each data object.

In our example, we defined that the myvector object of the Vector class should be produced from the MILEAGE column of the Cars table (see the StatObj table above). 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
<table>
<thead>
<tr>
<th>OBJECT</th>
<th>Dataset</th>
<th>Title</th>
<th>LIBRARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>cars</td>
<td>cars</td>
<td>Automobile Data Table</td>
<td>automob</td>
</tr>
</tbody>
</table>
```
This definition comprehensively describes the Cars table: its physical location and 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 corresponding to the Cars table exists. If not, an empty data set corresponding to the definitions in the Object and Property tables will be created.

Let’s consider one more example. Assume 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 of all, let’s define this new object in the StatObj table (see the highlighted rows):

The highlighted rows define a new object of the Vector Discrete class. The name of the object is newvector. For the simplicity, let’s consider only two parameters of the Vector Discrete class: name of the table, and name of the column that produces 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 new newvector object, you should add to the StatAct table the following definition (see the highlighted rows):

The definition stored in the third row on 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 (see the highlighted row):
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 implemented through a data dictionary (or 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, in turn, calls appropriate methods for each object that it processes. For example, the

\%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.

EXAMPLE

Let’s consider the following 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, 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 ;
```

In such a case this 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 %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=newvector, action="mean");
```

then %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):

<table>
<thead>
<tr>
<th>StatObj table</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATObj</td>
</tr>
<tr>
<td>myvector</td>
</tr>
<tr>
<td>myvector</td>
</tr>
<tr>
<td>newvector</td>
</tr>
<tr>
<td>newvector</td>
</tr>
<tr>
<td>r_vector</td>
</tr>
<tr>
<td>r_vector</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>StatAct table</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysid</td>
</tr>
<tr>
<td>example1</td>
</tr>
<tr>
<td>example1</td>
</tr>
<tr>
<td>example1</td>
</tr>
<tr>
<td>example1</td>
</tr>
</tbody>
</table>

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

DEVELOPING STATISTICAL MODELS

All statistical analysis has a model that attempts to describe the structure or relationships in some objects on which data is taken. Modern data analysis and data mining provide 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/data miners 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 which imitates, as closely as possible, the properties of the real objects being modeled. The differences between model and reality, the residuals, often are the key to reaching deeper understanding and a better model. Creating a statistical model usually involves the following steps:
1. Determine the variables to observe.
2. Collect and record data observations.
3. Study graphics and summaries of the collected data to discover and remove mistakes and to reveal low-dimensional relationships among variables.
4. Choose a model describing the important relationships seen or hypothesized in the data.
5. Fit the model using the appropriate modeling technique.
6. Examine the fit using model summaries and diagnostic plots.
7. Test the model with new data.
8. Repeat steps 4-7 until you are satisfied with the model.

The following flow chart visualizes modeling process:
1. Determine the variables to observe.

2. Collect and record data observations.

3. Study graphics and summaries of the collected data to discover and remove mistakes and to reveal low-dimensional relationships among variables.

4. Choose a model describing the important relationships seen or hypothesized in the data.

5. Fit the model using the appropriate modeling technique.

6. Examine the fit using model summaries and diagnostic plots.

7. Test the model with new data.

Are you satisfied with the model?

No

Yes

Figure 1. Flow chart of modeling process.

At any point in the modeling process described above, 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 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).

Roughly speaking, there is no one answer on 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 much more. Fortunately, many different classes of statistical models share a substantial common structure. The steps listed above 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 as 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. 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.

The types of data you have when developing a model 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 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 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 a level of data measurement or measurement scale. There are four different scales:

1. Nominal scale: classify each observation according to some characteristics. For example, sex (male/female), political party (Democrat/Republican/Independent/Other), binary measures (success/failure).
2. Ordinal scale: classify each observation according to some characteristic, but with some ordering. For example, number of cigarettes smoked (0, <1 pack/day, 1--5 packs/day, > 5 packs/day), feelings on an issue (disagree, indifferent, agree), military rank (Captain, Major, Colonel, General).
3. Interval scale: ordering inherent in the data, and even more importantly, there is a common and constant unit which is used for the measurement. Zero is not important. For example, temperature (Celsius, Fahrenheit, but not Kelvin).
4. Ratio scale: ratios are meaningful, as well 50K is half of 100K. For example, temperature on Kelvin scale, monetary units, weight, distance, 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 the data set, which contains a variety of data for car models. Here we just show you how this data set looks:
Below we show you how to define the statistical analysis of this data using an object-oriented approach. First of all, let’s update information about the Cars data object in the Property table. Such an update looks like this:

The P values of ATTRIBUT column of the Property table define that the COUNTRY and TYPE columns of the Cars table form the 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 definition of the new statistical objects looks in the StatObj table like this:
As you see, the definition 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. Let's consider the following definition of statistical analysis:

<table>
<thead>
<tr>
<th>StatAct table</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
<tr>
<td>example2</td>
</tr>
</tbody>
</table>

Now the actions will be performed according to the definition 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.

**CONCLUSION**

In this article 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 data analysis and data mining, and how it can be implemented in SAS using table-driven environment. Briefly, the idea is that:

- there are specially-structured tables forming a data dictionary and enabling you to define statistical classes, to define instances of these classes - that is, statistical objects, and to define statistical processing of these instances, and
- there is a mechanism implemented with SAS® macro language that resolves the definitions stored in the data dictionary.
REFERENCES


CONTACT INFORMATION

Tanya Kolosova
YieldWise.Com Canada Inc
6A-49 The Donway West Suite 918
Toronto, Ontario, Canada M3C 2E8
Phone: 416 841 0791
Email: tanya@watchwise.com

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.

APPENDIX

The main technique that is implemented by the %ACTION macro program is the evaluation of the data dictionary tables. The following is the code of this macro program:

```sas
/*
 * PROGRAM ACTION
 * DESCRIPTION The macro implementing methods invocation mechanism.
 * USAGE %action(object = object, action = action, params = params);
 * PARAMETERS REQUIRES The &libref macro variable is a global macro variable that contains the name of the SAS library where the data dictionary tables are located. The &libwork macro variable is a global macro variable that contains the name of the SAS library where the created statistical objects are located. In the book this library called _SA_WORK
 * AUTHORS T.Kolosova and S.Berestizhevsky.
 */
%macro action(object = object, action = action, params = params);
%let class = ;
%let method = ;
%let status = 0 ;
%let dsexist = 0 ;
%let action = %substr(%bquote(&action), 2, %eval(%length(&action)-2)) ;
data null ;
  set &libref..StatObj (where = (upcase(statobj) = upcase("&object"))) ;
  call symput("class", class) ;
run ;

data null ;
  set &libref..ClassMet (where = (upcase(class) = upcase("&class") and upcase(action) = upcase("&action"))) ;
  call symput("method", method) ;
*/
```
run;

data _null_;  
set &libref..Status (where =  
   (upcase(statobj) = upcase("&object") and  
   upcase(class) = upcase("&class")));  
call symput("status", 1);  
run;

%if &status = 1 %then  
%do;  
   proc sql noprint;  
      select count(*)  
         into : dsexist from dictionary.members  
            where (libname ? upcase("&libwork"))  
            & (memname ? upcase("&object"))  
            & (memtype ? "DATA");  
   quit;

   %if &dsexist = 0 %then  
      %let object = NULL ;  
%end;  
%if &status = 0 %then  
   %let object = NULL ;

   %&method (&object, %substr(%bquote(&params), 2,  
      %eval(%length(&params)-2))) ;  
%mend ;