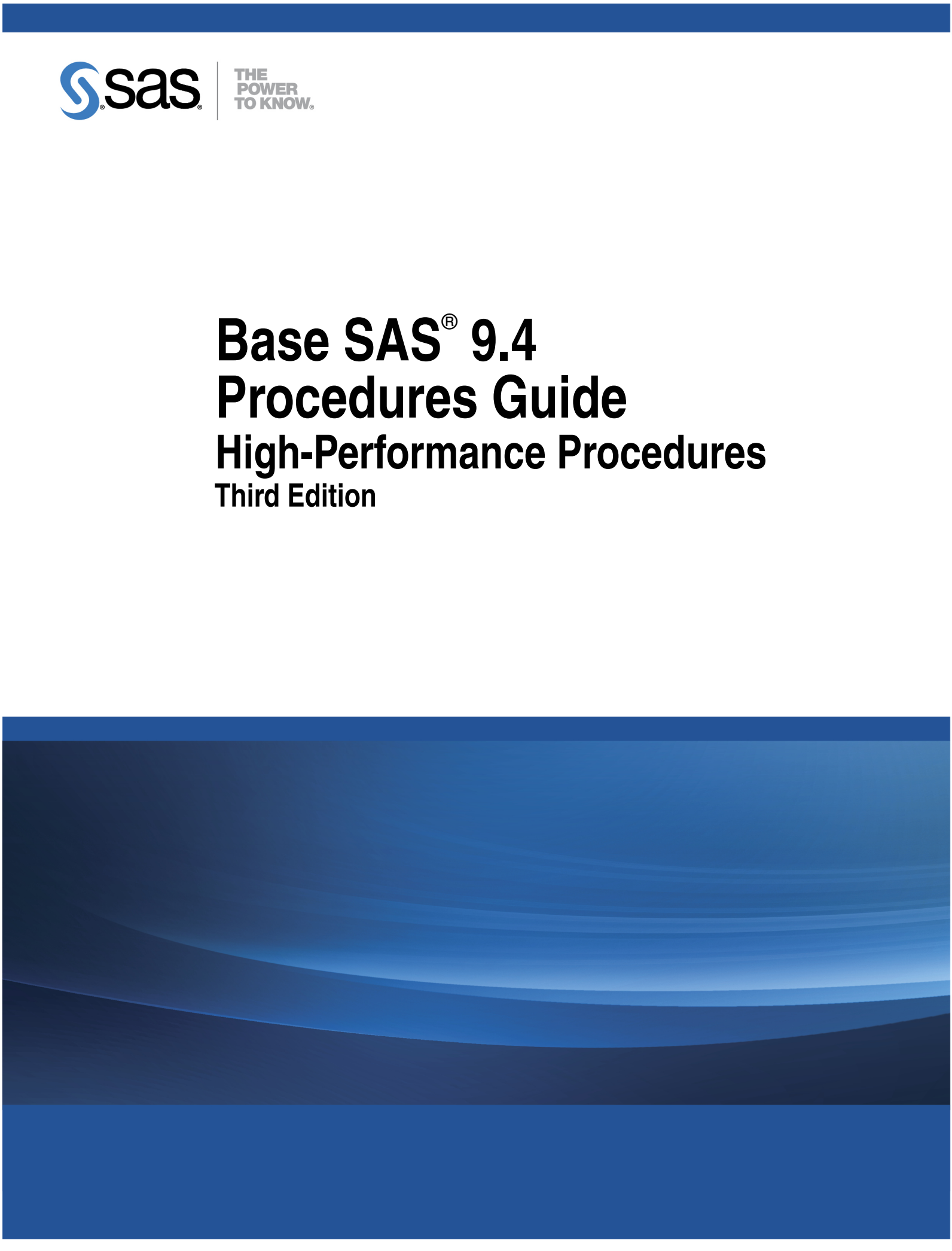


Base SAS® 9.4 Procedures Guide

High-Performance Procedures

Third Edition

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2014. *Base SAS® 9.4 Procedures Guide: High-Performance Procedures, Third Edition*. Cary, NC: SAS Institute Inc.

Base SAS® 9.4 Procedures Guide: High-Performance Procedures, Third Edition

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

August 2014

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

Contents

Credits and Acknowledgments

Credits

Documentation

Editing	Anne Baxter, Ed Huddleston
Documentation Support	Tim Arnold, Melanie Gratton

Software

The SAS High-Performance Analytics Server was implemented by the following members of the development staff. Program development includes design, programming, debugging, support, and documentation. In the following list, the names of the developers who currently provide primary support are listed first; other developers and previous developers are also listed.

HPBIN	Tao Wang, Taiping He
HPCORR	Charles Shorb
HPDMDB	Scott Pope
HPDS2	Mark Freskos, Tom P. Weber, Oliver Schabenberger
HPIMPUTE	Tao Wang
HPSAMPLE	Ye Liu
HPSUMMARY	Scott Mebust, Gordon Keener
High-performance computing foundation	Steve E. Krueger
High-performance analytics foundation	Robert Cohen, Georges H. Guirguis, Trevor Kearney, Richard Knight, Gang Meng, Scott Pope, Oliver Schabenberger, Charles Shorb, Tom P. Weber
Numerical routines	Georges H. Guirguis, Scott Pope

The following people contribute to the SAS High-Performance Analytics Server with their leadership and support: Chris Bailey, David Pope, Oliver Schabenberger, Renee Sciortino, Jonathan Wexler.

Testing

Tim Carter, Enzo D'Andreti, Alex Fang, Girija Gavankar, Dright Ho, Seungho Huh, Nilesh Jakhotiya, Cheryl LeSaint, Catherine Lopes, Jim McKenzie, Jim Metcalf, Huiping Miao, Phil Mohr, Bengt Pederson, Kim Sherrill, Weihua Shi, Stephanie Tysinger

Internationalization Testing

Feng Gao, Alex (Wenqi) He, David Li, Haiyong Rong, Bin Sun, Frank (Jidong) Wang, Lina Xu

Technical Support

Phil Gibbs

Acknowledgments

Many people make significant and continuing contributions to the development of SAS software products.

The final responsibility for the SAS System lies with SAS alone. We hope that you will always let us know your opinions about the SAS System and its documentation. It is through your participation that SAS software is continuously improved.

Chapter 1
What's New in Base SAS High-Performance Procedures 13.2

Contents

Overview

Infrastructure enhancements have been made and new functionality has been added in several procedures.

Infrastructure Enhancements

GRIDDATASERVER= Option in the PERFORMANCE Statement No Longer Needed

When you run high-performance analytical procedures alongside databases, you no longer need to specify the GRIDDATASERVER= option in the PERFORMANCE statement or set the GRIDDATASERVER environment variable. Any values that you specify are ignored.

GRIDMODE= Option in the PERFORMANCE Statement No Longer Needed

With one exception, which is described in the GRIDMODE= section of the PERFORMANCE statement documentation in Chapter 3, "Shared Concepts and Topics," you no longer need to specify the GRIDMODE=

option in the PERFORMANCE statement or set the GRIDMODE environment variable. Any values that you specify are ignored.

Multiple Data Access Modes Supported in a Single Procedure Step

The modes in which data sets are accessed are determined automatically for each data set that is used in a single procedure step.

Running Asymmetrically Alongside SAP HANA

High-performance analytical procedures can now read data from and write data to SAP HANA asymmetrically.

Procedure Enhancements

HPBIN Procedure

The HPBIN procedure now supports quantile binning.

HPCORR Procedure

The HPCORR procedure now supports the WITH statement. The WITH statement lists variables with which correlations of the VAR statement variables are to be computed.

HPSAMPLE Procedure

The HPSAMPLE procedure now supports a partitioning of the input data into three parts when you specify SAMPPCT2= along with the SAMPPCT= and PARTITION options. PROC HPSAMPLE now also supports up to four target variables for stratified sampling.

Chapter 2
Introduction

Contents

Overview of Base SAS High-Performance Procedures

This book describes Base SAS high-performance utility procedures that are included when you install one or more of the traditional SAS products that are shown in the following table:

Traditional Product	SAS High-Performance Analytics Product
SAS/STAT	SAS High-Performance Statistics
SAS/ETS	SAS High-Performance Econometrics
SAS/OR	SAS High-Performance Optimization
SAS High-Performance Forecasting	SAS High-Performance Forecasting
SAS Enterprise Miner	SAS High-Performance Data Mining
SAS Text Miner	SAS High-Performance Text Mining

You can run high-performance utility procedures in single-machine mode without licensing any of the SAS High-Performance Analytics products that are listed in the preceding table. However, to run these procedures in distributed mode, you must license at least one of the High-Performance Analytics products. For more information about single-machine mode and distributed mode, see Chapter 3, "Shared Concepts and Topics."

About This Book

This book assumes that you are familiar with Base SAS software and with the books *SAS Language Reference: Concepts* and *Base SAS Procedures Guide*. It also assumes that you are familiar with basic SAS System concepts, such as using the DATA step to create SAS data sets and using Base SAS procedures (such as the PRINT and SORT procedures) to manipulate SAS data sets.

Chapter Organization

This book is organized as follows:

Chapter 2, this chapter, provides an overview of high-performance utility procedures.

Chapter 3, "Shared Concepts and Topics," describes the modes in which SAS high-performance utility procedures can execute.

Subsequent chapters describe the high-performance utility procedures. These chapters appear in alphabetical order by procedure name. Each chapter is organized as follows:

- The "Overview" section provides a brief description of the functinality provided by the procedure.
- The "Getting Started" section provides a quick introduction to the procedure through a simple example.
- The "Syntax" section describes the SAS statements and options that control the procedure.
- The "Details" section discusses methodology and other topics, such as ODS tables.
- The "Examples" section contains examples that use the procedure.
- The "References" section contains references for the methodology.

Typographical Conventions

This book uses several type styles for presenting information. The following list explains the meaning of the typographical conventions used in this book:

roman	is the standard type style used for most text.
UPPERCASE ROMAN	is used for SAS statements, options, and other SAS language elements when they appear in the text. However, you can enter these elements in your own SAS programs in lowercase, uppercase, or a mixture of the two.
UPPERCASE BOLD	is used in the "Syntax" sections' initial lists of SAS statements and options.
oblique	is used in the syntax definitions and in text to represent arguments for which you supply a value.
VariableName	is used for the names of variables and data sets when they appear in the text.
bold	is used to for matrices and vectors.
italic	is used for terms that are defined in the text, for emphasis, and for references to publications.
`monospace`	is used for example code. In most cases, this book uses lowercase type for SAS code.

Options Used in Examples

The HTMLBLUE style is used to create the graphs and the HTML tables that appear in the online documentation. The PEARLJ style is used to create the PDF tables that appear in the documentation. A style template controls stylistic elements such as colors, fonts, and presentation attributes. You can specify a style template in an ODS destination statement as follows:

```
ods html style=HTMLBlue;
. . .
ods html close;

ods pdf style=PearlJ;
. . .
ods pdf close;
```

Most of the PDF tables are produced by using the following SAS System option:

```
options papersize=(6.5in 9in);
```

If you run the examples, you might get slightly different output. This is a function of the SAS System options that are used and the precision that your computer uses for floating-point calculations.

Online Documentation

This documentation is available online with the SAS System. To access SAS High-Performance Statistics software documentation from the SAS windowing environment, select **Help** from the main menu and then select **SAS Help and Documentation**. On the **Contents** tab, expand the **SAS Products**, **Base SAS**, and **Base SAS Procedures Guide: High-Performance Procedures** items. Then expand chapters and click on sections. You can search the documentation by using the **Search** tab.

You can also access the documentation by going to `http://support.sas.com/documentation`.

SAS Technical Support Services

The SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of the high- performance utility procedures. Go to `http://support.sas.com/techsup` for more information.

Chapter 3
Shared Concepts and Topics

Contents

Overview

This chapter describes the modes of execution in which SAS high-performance analytical procedures can execute. If you have Base SAS installed, you can run any procedure in this book on a single machine. However, to run procedures in this book in distributed mode, you must also have SAS High-Performance Server Distributed Mode software installed. For more information about these modes, see the next section.

This chapter provides details of how you can control the modes of execution and includes the syntax for the PERFORMANCE statement, which is common to all high-performance analytical procedures.

Processing Modes

Single-Machine Mode

Single-machine mode is a computing model in which multiple processors or multiple cores are controlled by a single operating system and can access shared resources, such as disks and memory. In this book, single-machine mode refers to an application running multiple concurrent threads on a multicore machine in order to take advantage of parallel execution on multiple processing units. More simply, single-machine mode for high-performance analytical procedures means multithreading on the client machine.

All high-performance analytical procedures are capable of running in single-machine mode, and this is the default mode when a procedure runs on the client machine. The procedure uses the number of CPUs (cores) on the machine to determine the number of concurrent threads. High-performance analytical procedures use different methods to map core count to the number of concurrent threads, depending on the analytic task. Using one thread per core is not uncommon for the procedures that implement data-parallel algorithms.

Distributed Mode

Distributed mode is a computing model in which several nodes in a distributed computing environment participate in the calculations. In this book, the distributed mode of a high-performance analytical procedure refers to the procedure performing the analytics on an appliance that consists of a cluster of nodes. This appliance can be one of the following:

- a database management system (DBMS) appliance on which the SAS High-Performance Analytics infrastructure is also installed
- a cluster of nodes that have the SAS High-Performance Analytics infrastructure installed but no DBMS software installed

Controlling the Execution Mode with Environment Variables and Performance Statement Options

You control the execution mode by using environment variables or by specifying options in the PERFORMANCE statement in high-performance analytical procedures, or by a combination of these methods.

The important environment variables follow:

- *grid host* identifies the domain name system (DNS) or IP address of the appliance node to which the SAS High-Performance Server Distributed Mode software connects to run in distributed mode.
- *installation location* identifies the directory where the SAS High-Performance Server Distributed Mode software is installed on the appliance.

You can set an environment variable directly from the SAS program by using the OPTION SET= command. For example, the following statements define the grid host and the location where the SAS High-Performance software is installed on the appliance:

```
option set=GRIDHOST      ="hpa.sas.com";
option set=GRIDINSTALLLOC="/opt/TKGrid";
```

Alternatively, you can set the parameters in the PERFORMANCE statement in high-performance analytical procedures. For example:

```
performance host      ="hpa.sas.com"
            install   ="/opt/TKGrid";
```

A specification in the PERFORMANCE statement overrides a specification of an environment variable without resetting its value. An environment variable that you set in the SAS session by using an OPTION SET= command remains in effect until it is modified or until the SAS session terminates.

The key variable that determines whether a high-performance analytical procedure executes in single-machine or distributed mode is the *grid host*. The installation location is needed to ensure that a connection to the grid host can be made, given that a host is specified. This book assumes that the installation location has been set by your system administrator.

The following sets of SAS statements are functionally equivalent:

```
proc hpreduce;
   reduce unsupervised x:;
   performance host="hpa.sas.com";
run;

option set=GRIDHOST="hpa.sas.com";
proc hpreduce;
   reduce unsupervised x:;
run;
```

Determining Single-Machine Mode or Distributed Mode

High-performance analytical procedures use the following rules to determine whether they run in single-machine mode or distributed mode:

- If a grid host is not specified, the analysis is carried out in single-machine mode on the client machine that runs the SAS session.
- If a grid host is specified, the behavior depends on whether the execution is alongside the database or alongside HDFS. If the data are local to the client (that is, not stored in the distributed database or HDFS on the appliance), you need to use the NODES= option in the PERFORMANCE statement to specify the number of nodes on the appliance or cluster that you want to engage in the analysis. If the procedure executes alongside the database or alongside HDFS, you do not need to specify the NODES= option.

The following example shows single-machine and client-data distributed configurations for a data set of 100,000 observations that are simulated from a logistic regression model. The following DATA step generates the data:

```
data simData;
   array _a{8} _temporary_ (0,0,0,1,0,1,1,1);
   array _b{8} _temporary_ (0,0,1,0,1,0,1,1);
   array _c{8} _temporary_ (0,1,0,0,1,1,0,1);
   do obsno=1 to 100000;
      x  = rantbl(1,0.28,0.18,0.14,0.14,0.03,0.09,0.08,0.06);
      a  = _a{x};
      b  = _b{x};
      c  = _c{x};
      x1 = int(ranuni(1)*400);
      x2 = 52 + ranuni(1)*38;
      x3 = ranuni(1)*12;
      lp = 6. -0.015*(1-a) + 0.7*(1-b) + 0.6*(1-c) + 0.02*x1 -0.05*x2 - 0.1*x3;
      y  = ranbin(1,1,(1/(1+exp(lp))));
      output;
   end;
   drop x lp;
run;
```

The following statements run PROC HPLOGISTIC to fit a logistic regression model:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
run;
```

Figure 3.1 shows the results from the analysis.

Figure 3.1 Results from Logistic Regression in Single-Machine Mode

The HPLOGISTIC Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Data Access Information			
Data	**Engine**	**Role**	**Path**
WORK.SIMDATA	V9	Input	On Client

Model Information	
Data Source	WORK.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

The entries in the "Performance Information" table show that the HPLOGISTIC procedure runs in single-machine mode and uses four threads, which are chosen according to the number of CPUs on the client machine. You can force a certain number of threads on any machine that is involved in the computations by specifying the NTHREADS option in the PERFORMANCE statement. Another indication of execution on the client is the following message, which is issued in the SAS log by all high-performance analytical procedures:

```
NOTE: The HPLOGISTIC procedure is executing in single-machine mode.
```

The following statements use 10 nodes (in distributed mode) to analyze the data on the appliance; results appear in Figure 3.2:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host="hpa.sas.com" nodes=10;
run;
```

Figure 3.2 Results from Logistic Regression in Distributed Mode

The HPLOGISTIC Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	10
Number of Threads per Node	24

The specification of a host causes the "Performance Information" table to display the name of the host node of the appliance. The "Performance Information" table also indicates that the calculations were performed in a distributed environment on the appliance. Twenty-four threads on each of 10 nodes were used to perform the calculations—for a total of 240 threads.

Another indication of distributed execution on the appliance is the following message, which is issued in the SAS log by all high-performance analytical procedures:

```
NOTE: The HPLOGISTIC procedure is executing in the distributed
      computing environment with 10 worker nodes.
```

You can override the presence of a grid host and force the computations into single-machine mode by specifying the NODES=0 option in the PERFORMANCE statement:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host="hpa.sas.com" nodes=0;
run;
```

Figure 3.3 shows the "Performance Information" table. The numeric results are not reproduced here, but they agree with the previous analyses, which are shown in Figure 3.1 and Figure 3.2.

Figure 3.3 Single-Machine Mode Despite Host Specification

The HPLOGISTIC Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Data Access Information			
Data	**Engine**	**Role**	**Path**
WORK.SIMDATA	V9	Input	On Client

The "Performance Information" table indicates that the HPLOGISTIC procedure executes in single-machine mode on the client. This information is also reported in the following message, which is issued in the SAS log:

```
NOTE: The HPLOGISTIC procedure is executing in single-machine mode.
```

In the analysis shown previously in Figure 3.2, the data set Work.simData is local to the client, and the HPLOGISTIC procedure distributed the data to 10 nodes on the appliance. The High-Performance Analytics infrastructure does not keep these data on the appliance. When the procedure terminates, the in-memory representation of the input data on the appliance is freed.

When the input data set is large, the time that is spent sending client-side data to the appliance might dominate the execution time. In practice, transfer speeds are usually lower than the theoretical limits of the network connection or disk I/O rates. At a transfer rate of 40 megabytes per second, sending a 10-gigabyte data set to the appliance requires more than four minutes. If analytic execution time is in the range of seconds, the "performance" of the process is dominated by data movement.

The alongside-the-database execution model, unique to high-performance analytical procedures, enables you to read and write data in distributed form from the database that is installed on the appliance.

Data Access Modes

Single-Machine Data Access Mode

When high-performance analytical procedures run in single-machine mode, they access data in the same way as traditional SAS procedures. They use Base SAS to access input and output SAS data sets on the client machine, and they use the relevant SAS/ACCESS interface to bring data from other sources, such as third-party databases, Hadoop, and SAS LASR servers, to the client.

Distributed Data Access Mode

When high-performance analytical procedures run in distributed mode, input data must be brought to the computation that is performed on the nodes of the grid, and output data must be sent from the computational nodes. This can be accomplished in several ways:

- Client-data (local-data) mode: The input and output data for the analytic task are stored on the client machine where the high-performance procedure is invoked. When the procedure runs, the SAS High-Performance Analytics infrastructure sends input data from the client to the distributed computing environment and sends output data from the distributed computing environment to the client.
- Parallel symmetric mode: Input and output data are stored on the same nodes that are used for the distributed computation, and the data move in parallel from the data store to the computational nodes without crossing node boundaries. Parallel symmetric mode is available with the following distributed data sources:
  - Data in Greenplum or Teradata databases that are collocated with the computational nodes. This access mode is also called alongside-the-database mode. For more information, see the section "Alongside-the-Database Execution" on page 18.

  - Data in SASHDAT format in the Hadoop Distributed File System (HDFS) that is collocated with the computational nodes. This access mode is also called alongside-HDFS mode. For more information, see the section "Alongside-HDFS Execution by Using the SASHDAT Engine" on page 29.
  - Data in a SAS LASR Analytic Server that is collocated with the computational nodes. This access mode is also called alongside-LASR mode. For more information, see the section "Running High-Performance Analytical Procedures Alongside a SAS LASR Analytic Server in Distributed Mode" on page 20.

- Parallel asymmetric mode: The primary reason for providing this mode is to enable you to manage and house data on appliances (the data appliances) and to run high-performance analytical procedures on a different appliance (the computing appliance). The high-performance analytical procedures run in a SAS process on the computing appliance. For each data source that is accessed in parallel asymmetric mode, a SAS Embedded Process must run on the associated data appliance. Data are requested by a SAS data feeder that runs on the computing appliance and communicates with the SAS Embedded Process on the data appliance. The SAS Embedded Process transfers the data in parallel to the SAS data feeder that runs on each of the nodes of the computing appliance. This mode is called asymmetric mode because the number of nodes on the data appliance does not need to match the number of nodes on the computing appliance. Parallel asymmetric mode is supported for data in Teradata, Greenplum, and Oracle databases and for data in HDFS and SAP HANA. In these cases, the parallel asymmetric access is somewhat loosely described as being asymmetric alongside access, even though the data storage and computation can occur on different appliances. For more information, see the section "Running High-Performance Analytical Procedures in Asymmetric Mode" on page 23.

- Through-the-client mode: When data can be accessed through a SAS/ACCESS interface but the data reside in a file system or in a distributed data source on which a SAS Embedded Process is not running, those data cannot be accessed in parallel in either symmetric or asymmetric mode. The SAS/ACCESS interface is used to transfer input data from the data source to the client machine on which the high-performance procedure is invoked, and the data are then sent to the distributed computing environment by the SAS High-Performance Analytics infrastructure. The data path is reversed for output data. This mode of data access is referred to as through-the-client access.

Determining the Data Access Mode

High-performance analytical procedures determine the data access mode individually for each data set that is used in the analysis. When high-performance analytical procedures run in distributed mode, parallel symmetric or parallel asymmetric mode is used whenever possible. There are two reasons why parallel access might not be possible. The first reason is that for a particular data set, the required SAS Embedded Process is not installed on the appliance that houses the data. In such cases, access to those data reverts to through-the-client access, and a note like the following is reported in the SAS log:

```
NOTE: The data MYLIB.MYDATA are being routed through the client because a
      SAS Embedded Process is not running on the associated data server.
```

The second reason why parallel data access might not be possible for a particular data set is that the required driver software might not be installed on the compute nodes. In this case, the required data feeder that

moves the data from the compute nodes to the data source cannot be successfully loaded, and a note like the following is reported in the SAS log:

```
NOTE: The data MYLIB.MYDATA are being routed through the client because
      the ORACLE data feeder could not be loaded on the specified grid host.
```

For distributed data in SASHDAT format in HDFS or data in a SAS LASR Analytic Server, parallel symmetric access is used when the data nodes and compute nodes are collocated on the same appliance. For data in a LASR Analytic Server that cannot be accessed in parallel symmetric mode, through-the-client mode is used. Through-the-client access is not supported for data in SASHDAT format in HDFS.

For data in Greenplum databases, parallel symmetric access is used if the compute nodes and the data nodes are collocated on the same appliance and you do not specify the NODES=*n* option in a PERFORMANCE statement. In this case, the number of nodes that are used is determined by the number of nodes across which the data are distributed. If you specify NODES=*n*, then parallel asymmetric access is used.

For data in Teradata databases, parallel asymmetric mode is used by default. If your data and computation are collocated on the same appliance and you want to use parallel symmetric mode, then you need to specify GRIDMODE=SYM in the PERFORMANCE statement or you need to set the GRIDMODE='SYM' environment variable by using an OPTION SET statement.

High-performance analytical procedures produce a "Data Access Information" table that shows you how each data set that is used in the analysis is accessed. The following statements provide an example in which PROC HPDS2 is used to copy a distributed data set named Neuralgia (which is stored in SASHDAT format in HDFS) to a SAS data set on the client machine:

```
libname hdatlib sashdat
                host='hpa.sas.com';
                hdfs_path="/user/hps";

proc hpds2 data=hdatlib.neuralgia out=neuralgia;
    performance host='hpa.sas.com';
    data DS2GTF.out;
       method run();
      set DS2GTF.in;
    end;
   enddata;
 run;
```

Figure 3.4 shows the output that PROC HPDS2 produces. The "Performance Information" table shows that PROC HPDS2 ran in distributed mode on a 13-node grid. The "Data Access Information" table shows that the input data were accessed in parallel symmetric mode and the output data set was sent to the client, where the V9 (base) engine stored it as a SAS data set in the Work directory.

Figure 3.4 Performance Information and Data Access Information Tables

The HPDS2 Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	13
Number of Threads per Node	24

Figure 3.4 *continued*

Data Access Information			
Data	**Engine**	**Role**	**Path**
HDATLIB.NEURALGIA	SASHDAT	Input	Parallel, Symmetric
WORK.NEURALGIA	V9	Output	To Client

Alongside-the-Database Execution

High-performance analytical procedures interface with the distributed database management system (DBMS) on the appliance in a unique way. If the input data are stored in the DBMS and the grid host is the appliance that houses the data, high-performance analytical procedures create a distributed computing environment in which an analytic process is collocated with the nodes of the DBMS. Data then pass from the DBMS to the analytic process on each node. Instead of moving across the network and possibly back to the client machine, the data pass locally between the processes on each node of the appliance.

Because the analytic processes on the appliance are separate from the database processes, the technique is referred to as alongside-the-database execution in contrast to in-database execution, where the analytic code executes in the database process.

In general, when you have a large amount of input data, you can achieve the best performance from high-performance analytical procedures if execution is alongside the database.

Before you can run alongside the database, you must distribute the data to the appliance. The following statements use the HPDS2 procedure to distribute the data set Work.simData into the mydb database on the hpa.sas.com appliance. In this example, the appliance houses a Greenplum database.

```
option set=GRIDHOST="green.sas.com";
libname applianc greenplm
        server  ="green.sas.com"
        user    =XXXXXX
        password=YYYYY
        database=mydb;


proc datasets lib=applianc nolist; delete simData;
proc hpds2 data=simData
           out =applianc.simData(distributed_by='distributed randomly');
  performance commit=10000 nodes=all;
  data DS2GTF.out;
     method run();
        set DS2GTF.in;
     end;
  enddata;
run;
```

If the output table applianc.simData exists, the DATASETS procedure removes the table from the Greenplum database because a DBMS does not usually support replacement operations on tables.

Note that the libref for the output table points to the appliance. The data set option informs the HPDS2 procedure to distribute the records randomly among the data segments of the appliance. The statements that follow the PERFORMANCE statement are the DS2 program that copies the input data to the output data without further transformations.

Because you loaded the data into a database on the appliance, you can use the following HPLOGISTIC statements to perform the analysis on the appliance in the alongside-the-database mode. These statements are almost identical to the first PROC HPLOGISTIC example in a previous section, which executed in single-machine mode.

```
proc hplogistic data=applianc.simData;
   class a b c;
   model y = a b c x1 x2 x3;
run;
```

The subtle differences are as follows:

- The grid host environment variable that you specified in an OPTION SET= command is still in effect.
- The DATA= option in the high-performance analytical procedure uses a libref that identifies the data source as being housed on the appliance. This libref was specified in a prior LIBNAME statement.

Figure 3.5 shows the results from this analysis. The "Performance Information" table shows that the execution was in distributed mode, and the "Data Access Information" table shows that the data were accessed asymmetrically in parallel from the Greenplum database. The numeric results agree with the previous analyses, which are shown in Figure 3.1 and Figure 3.2.

Figure 3.5 Alongside-the-Database Execution on Greenplum

The HPLOGISTIC Procedure

Performance Information	
Host Node	green.sas.com
Execution Mode	Distributed
Number of Compute Nodes	16
Number of Threads per Node	24

Data Access Information			
Data	**Engine**	**Role**	**Path**
APPLIANC.SIMDATA	GREENPLM	Input	Parallel, Asymmetric

Model Information	
Data Source	APPLIANC.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Figure 3.5 *continued*

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

Alongside-LASR Distributed Execution

You can execute high-performance analytical procedures in distributed mode alongside a SAS LASR Analytic Server. When high-performance analytical procedures run in this mode, the data are preloaded in distributed form in memory that is managed by a LASR Analytic Server. The data on the nodes of the appliance are accessed in parallel in the process that runs the LASR Analytic Server, and they are transferred to the process where the high-performance analytical procedure runs. In general, each high-performance analytical procedure copies the data to memory that persists only while that procedure executes. Hence, when a high-performance analytical procedure runs alongside a LASR Analytic Server, both the high-performance analytical procedure and the LASR Analytic Server have a copy of the subset of the data that is used by the high-performance analytical procedure. The advantage of running high-performance analytical procedures alongside a LASR Analytic Server (as opposed to running alongside a DBMS table or alongside HDFS) is that the initial transfer of data from the LASR Analytic Server to the high-performance analytical procedure is a memory-to-memory operation that is faster than the disk-to-memory operation when the procedure runs alongside a DBMS or HDFS. When the cost of preloading a table into a LASR Analytic Server is amortized by multiple uses of these data in separate runs of high-performance analytical procedures, using the LASR Analytic Server can result in improved performance.

Running High-Performance Analytical Procedures Alongside a SAS LASR Analytic Server in Distributed Mode

This section provides an example of steps that you can use to start and load data into a SAS LASR Analytic Server instance and then run high-performance analytical procedures alongside this LASR Analytic Server instance.

Starting a SAS LASR Analytic Server Instance

The following statements create a SAS LASR Analytic Server instance and load it with the simData data set that is used in the preceding examples. The data that are loaded into the LASR Analytic Server persist in memory across procedure boundaries until these data are explicitly deleted or until the server instance is terminated.

```
proc lasr port=54545
          data=simData
          path="/tmp/";
   performance host="hpa.sas.com" nodes=ALL;
run;
```

The PORT= option specifies a network port number to use. The PATH= option specifies the directory in which the server and table signature files are to be stored. The specified directory must exist on each machine in the cluster. The DATA= option specifies the name of a data set that is loaded into this LASR Analytic Server instance. (You do not need to specify the DATA= option at this time because you can add tables to the LASR Analytic Server instance at any stage of its life.) For more information about starting and using a LASR Analytic Server, see the *SAS LASR Analytic Server: Reference Guide*.

The NODES=ALL option in the PERFORMANCE statement specifies that the LASR Analytic Server run on all the nodes on the appliance. You can start a LASR Analytic Server on a subset of the nodes on an appliance, but this might affect whether high-performance analytical procedures can run alongside the LASR Analytic Server. For more information, see the section "Alongside-LASR Distributed Execution on a Subset of the Appliance Nodes" on page 23.

Figure 3.6 shows the "Performance Information" and "Data Access Information" tables, which show that the LASR procedure ran in distributed mode on 13 nodes and that the data were sent from the client to the appliance.

Figure 3.6 Performance and Data Access Information

The LASR Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	13

Data Access Information			
Data	**Engine**	**Role**	**Path**
WORK.SIMDATA	V9	Input	From Client

Associating a SAS Libref with the SAS LASR Analytic Server Instance

The following statements use a LIBNAME statement that associates a SAS libref (named MyLasr) with tables on the server instance as follows:

```
libname MyLasr sasiola port=54545 host="hpa.sas.com";
```

The SASIOLA option requests that the MyLasr libref use the SASIOLA engine, and the PORT= value associates this libref with the appropriate server instance. For more information about creating a libref that uses the SASIOLA engine, see the *SAS LASR Analytic Server: Reference Guide*.

Running a High-Performance Analytical Procedure Alongside the SAS LASR Analytic Server Instance

You can use the MyLasr libref to specify the input data for high-performance analytical procedures. You can also create output data sets in the SAS LASR Analytic Server instance by using this libref to request that the output data set be held in memory by the server instance as follows:

```
proc hplogistic data=MyLasr.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   output out=MyLasr.simulateScores pred=PredictedProbabliity;
run;
```

Because you previously specified the GRIDHOST= environment variable and the input data are held in distributed form in the associated server instance, this PROC HPLOGISTIC step runs in distributed mode alongside the LASR Analytic Server, as indicated in the "Performance Information" table shown in Figure 3.7.

Figure 3.7 Performance and Data Access Information

The HPLOGISTIC Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	13
Number of Threads per Node	24

Data Access Information			
Data	**Engine**	**Role**	**Path**
MYLASR.SIMDATA	SASIOLA	Input	Parallel, Symmetric
MYLASR.SIMULATESCORES	SASIOLA	Output	Parallel, Symmetric

The "Data Access Information" table shows that both the input and output data were read and written, respectively, in parallel symmetric mode.

The preceding OUTPUT statement creates an output table that is added to the LASR Analytic Server instance. Output data sets do not have to be created in the same server instance that holds the input data. You can use a different LASR Analytic Server instance to hold the output data set. However, in order for the output data to be created in parallel symmetric mode, all the nodes that are used by the server instance that holds the input data must also be used by the server instance that holds the output data.

Terminating a SAS LASR Analytic Server Instance

You can continue to run high-performance analytical procedures and add and delete tables from the SAS LASR Analytic Server instance until you terminate the server instance as follows:

```
proc lasr term port=54545;
run;
```

Alongside-LASR Distributed Execution on a Subset of the Appliance Nodes

When you run PROC LASR to start a SAS LASR Analytic Server, you can specify the NODES= option in a PERFORMANCE statement to control how many nodes the LASR Analytic Server executes on. Similarly, a high-performance analytical procedure can execute on a subset of the nodes either because you specify the NODES= option in a PERFORMANCE statement or because you run alongside a DBMS or HDFS with an input data set that is distributed on a subset of the nodes on an appliance. In such situations, if a high-performance analytical procedure uses nodes on which the LASR Analytic Server is not running, then running alongside LASR is not supported. You can avoid this issue by specifying the NODES=ALL in the PERFORMANCE statement when you use PROC LASR to start the LASR Analytic Server.

Running High-Performance Analytical Procedures in Asymmetric Mode

This section provides examples of how you can run high-performance analytical procedures in asymmetric mode. It also includes examples that run in symmetric mode to highlight differences between the modes. For a description of asymmetric mode, see the section "Distributed Data Access Mode" on page 15.

Asymmetric mode is commonly used when the data appliance and the computing appliance are distinct appliances. In order to be able to use an appliance as a data provider for high-performance analytical procedures that run in asymmetric mode on another appliance, it is not necessary that SAS High-Performance Server Distributed Mode be installed on the data appliance. However, it is essential that a SAS Embedded Process be installed on the data appliance and that SAS High-Performance Server Distributed Mode be installed on the computing appliance.

The following examples use a 24-node data appliance named "data_appliance.sas.com," which houses a Teradata DBMS and has a SAS Embedded Process installed. Because SAS High-Performance Server Distributed Mode is also installed on this appliance, it can be used to run high-performance analytical procedures in both symmetric and asymmetric modes.

The following statements load the simData data set of the preceding sections onto the data appliance:

```
libname dataLib teradata
        server  ="tera2650"
        user    =XXXXXX
        password=YYYYY
        database=mydb;

data dataLib.simData;
   set simData;
run;
```

NOTE: You can provision the appliance with data even if SAS High-Performance Server Distributed Mode software is not installed on the appliance.

The following subsections show how you can run the HPLOGISTIC procedure symmetrically and asymmetrically on a single data appliance and asymmetrically on distinct data and computing appliances.

Running in Symmetric Mode

The following statements run the HPLOGISTIC procedure in symmetric mode on the data appliance:

```
proc hplogistic data=dataLib.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host       = "data_appliance.sas.com"
               nodes      = 10
               gridmode   = sym;
run;
```

Figure 3.8 shows the results of this analysis.

Figure 3.8 Alongside-the-Database Execution in Symmetric Mode on Teradata

The HPLOGISTIC Procedure

Performance Information	
Host Node	data_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	24
Number of Threads per Node	24

Data Access Information			
Data	**Engine**	**Role**	**Path**
DATALIB.simData	TERADATA	Input	Parallel, Symmetric

Figure 3.8 *continued*

Model Information	
Data Source	DATALIB.simData
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

The "Performance Information" table shows that the execution occurs in symmetric mode on the 24 nodes of the data appliance. In this case, the NODES=10 option in the PERFORMANCE statement is ignored because the number of nodes that are used is determined by the number of nodes across which the data are distributed, as indicated in the following warning message in the SAS log:

```
WARNING: The NODES=10 option in the PERFORMANCE statement is ignored because
         you are running alongside the distributed data source
         DATALIB.simData.DATA. The number of compute nodes is determined by the
         configuration of the distributed DBMS.
```

Running in Asymmetric Mode on One Appliance

By default, the HPLOGISTIC procedure runs in asymmetric mode, as shown Figure 3.9, which is produced by the following statements:

```
proc hplogistic data=dataLib.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host        = "data_appliance.sas.com"
               nodes       = 10;
run;
```

Figure 3.9 Alongside-Teradata Execution in Asymmetric Mode

The HPLOGISTIC Procedure

Performance Information	
Host Node	data_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	10
Number of Threads per Node	24

Data Access Information			
Data	**Engine**	**Role**	**Path**
DATALIB.simData	TERADATA	Input	Parallel, Asymmetric

The "Performance Information" table confirms that the NODES=10 option that you specified in the PERFORMANCE statement was honored, and the "Data Access Information" table shows that the data were accessed in parallel asymmetric mode. The data were moved in parallel from the 24 nodes on which the data were stored to the 10 nodes on which the execution occurred. The numeric results are not reproduced here, but they agree with the previous analyses.

Running in Asymmetric Mode on Distinct Appliances

Usually, there is no advantage to executing high-performance analytical procedures in asymmetric mode on one appliance, because data might have to be unnecessarily moved between nodes. The following example demonstrates the more typical use of asymmetric mode. In this example, the specified grid host "compute_appliance.sas.com" is a 142-node computing appliance that is different from the 24-node data appliance "data_appliance.sas.com," which houses the Teradata DBMS where the data reside.

The advantage of using different computing and data appliances is that the data appliance is not affected by the execution of high-performance analytical procedures except during the initial parallel data transfer. A potential disadvantage of this asymmetric mode of execution is that the performance can be limited by the bandwidth with which data can be moved between the appliances. However, because this data movement takes place in parallel from the nodes of the data appliance to the nodes of the computing appliance, this potential performance bottleneck can be overcome with appropriately provisioned hardware. The following statements show how this is done:

```
proc hplogistic data=dataLib.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host = "compute_appliance.sas.com" nodes=30;
run;
```

Figure 3.10 shows the "Performance Information" and "Data Access Information" tables.

Figure 3.10 Asymmetric Mode with Distinct Data and Computing Appliances

The HPLOGISTIC Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	30
Number of Threads per Node	32

Data Access Information			
Data	**Engine**	**Role**	**Path**
DATALIB.simData	TERADATA	Input	Parallel, Asymmetric

PROC HPLOGISTIC ran on 30 nodes of the computing appliance, even though the data were partitioned across the 24 nodes of the data appliance. The numeric results are not reproduced here, but they agree with the previous analyses shown in Figure 3.1 and Figure 3.2.

Every time you run a high-performance analytical procedure in asymmetric mode that uses different computing and data appliances, data are transferred between these appliances. If you plan to make repeated use of the same data, then it might be advantageous to temporarily persist the data that you need on the computing appliance. One way to persist the data is to store them as a table in a SAS LASR Analytic Server that runs on the computing appliance. By running PROC LASR in asymmetric mode, you can load the data in parallel from the data appliance nodes to the nodes on which the LASR Analytic Server runs on the computing appliance. You can then use a LIBNAME statement that associates a SAS libref with tables on the LASR Analytic Server. The following statements show how you do this:

```
proc lasr port=54345
          data=dataLib.simData
          path="/tmp/";
   performance host ="compute_appliance.sas.com" nodes=30;
run;

libname MyLasr sasiola tag="dataLib" port=54345 host="compute_appliance.sas.com" ;
```

Figure 3.11 show the "Performance Information" and "Data Access Information" tables.

Figure 3.11 PROC LASR Running in Asymmetric Mode

The LASR Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	30

Data Access Information			
Data	**Engine**	**Role**	**Path**
DATALIB.simData	TERADATA	Input	Parallel, Asymmetric

By default, all the nodes on the computing appliance would be used. However, because NODES=30 was specified in the PERFORMANCE statement, PROC LASR ran on only 30 nodes of the computing appliance.

The data were loaded asymmetrically in parallel from the 24 data appliance nodes to the 30 compute nodes on which PROC LASR ran.

After the data are loaded into a LASR Analytic Server that runs on the computing appliance, you can run high-performance analytical procedures alongside this LASR Analytic Server as shown by the following statements:

```
proc hplogistic data=MyLasr.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   output out=MyLasr.myOutputData pred=myPred;
   performance host = "compute_appliance.sas.com";
run;
```

The following note, which appears in the SAS log, confirms that the output data set is created successfully:

```
NOTE: The table DATALIB.MYOUTPUTDATA has been added to the LASR Analytic Server
      with port 54345. The Libname is MYLASR.
```

You can use the dataLib libref that you used to load the data onto the data appliance to create an output data set on the data appliance.

```
proc hplogistic data=MyLasr.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   output out=dataLib.myOutputData pred=myPred;
   performance host  = "compute_appliance.sas.com";
run;
```

The following note, which appears in the SAS log, confirms that the output data set is created successfully on the data appliance:

```
NOTE: The data set DATALIB.myOutputData has 100000 observations and 1 variables.
```

When you run a high-performance analytical procedure on a computing appliance and either read data from or write data to a different data appliance on which a SAS Embedded Process is running, the Read and Write operations take place in parallel without any movement of data to and from the SAS client.

When you no longer need the data in the SAS LASR Analytic Server, you should terminate the server instance as follows:

```
proc lasr term port=54345;
    performance host="compute_appliance.sas.com";
run;
```

If you configured Hadoop on the computing appliance, then you can create output data tables that are stored in the HDFS on the computing appliance. You can do this by using the SASHDAT engine as described in the section “Alongside-HDFS Execution” on page 29.

Alongside-HDFS Execution

Running high-performance analytical procedures alongside HDFS shares many features with running alongside the database. You can execute high-performance analytical procedures alongside HDFS by using either the SASHDAT engine or the Hadoop engine.

You use the SASHDAT engine to read and write data that are stored in HDFS in a proprietary SASHDAT format. In SASHDAT format, metadata that describe the data in the Hadoop files are included with the data. This enables you to access files in SASHDAT format without supplying any additional metadata. Additionally, you can also use the SASHDAT engine to read data in CSV (comma-separated value) format, but you need supply metadata that describe the contents of the CSV data. The SASHDAT engine provides highly optimized access to data in HDFS that are stored in SASHDAT format.

The Hadoop engine reads data that are stored in various formats from HDFS and writes data to HDFS in CSV format. This engine can use metadata that are stored in Hive, which is a data warehouse that supplies metadata about data that are stored in Hadoop files. In addition, this engine can use metadata that you create by using the HDMD procedure.

The following subsections provide details about using the SASHDAT and Hadoop engines to execute high-performance analytical procedures alongside HDFS.

Alongside-HDFS Execution by Using the SASHDAT Engine

If the grid host is a cluster that houses data that have been distributed by using the SASHDAT engine, then high-performance analytical procedures can analyze those data in the alongside-HDFS mode. The procedures use the distributed computing environment in which an analytic process is collocated with the nodes of the cluster. Data then pass from HDFS to the analytic process on each node of the cluster.

Before you can run a procedure alongside HDFS, you must distribute the data to the cluster. The following statements use the SASHDAT engine to distribute to HDFS the simData data set that was used in the previous two sections:

```
option set=GRIDHOST="hpa.sas.com";

libname hdatLib sashdat
        path="/hps";

 data hdatLib.simData (replace = yes) ;
     set simData;
 run;
```

In this example, the GRIDHOST is a cluster where the SAS Data in HDFS Engine is installed. If a data set that is named simData already exists in the hps directory in HDFS, it is overwritten because the REPLACE=YES data set option is specified. For more information about using this LIBNAME statement, see the section "LIBNAME Statement for the SAS Data in HDFS Engine" in the *SAS LASR Analytic Server: Reference Guide*.

The following HPLOGISTIC procedure statements perform the analysis in alongside-HDFS mode. These statements are almost identical to the PROC HPLOGISTIC example in the previous two sections, which executed in single-machine mode and alongside-the-database distributed mode, respectively.

Figure 3.12 shows the "Performance Information" and "Data Access Information" tables. You see that the procedure ran in distributed mode and that the input data were read in parallel symmetric mode. The numeric results shown in Figure 3.13 agree with the previous analyses shown in Figure 3.1, Figure 3.2, and Figure 3.5.

Figure 3.12 Alongside-HDFS Execution Performance Information

The HPLOGISTIC Procedure

Performance Information	
Host Node	hpa.sas.com
Execution Mode	Distributed
Number of Compute Nodes	13
Number of Threads per Node	24

Data Access Information			
Data	**Engine**	**Role**	**Path**
HDATLIB.SIMDATA	SASHDAT	Input	Parallel, Symmetric

Figure 3.13 Alongside-HDFS Execution Model Information

Model Information	
Data Source	HDATLIB.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates					
Parameter	**Estimate**	**Standard Error**	**DF**	**t Value**	**Pr > |t|**
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

Alongside-HDFS Execution by Using the Hadoop Engine

The following LIBNAME statement sets up a libref that you can use to access data that are stored in HDFS and have metadata in Hive:

```
libname hdoopLib hadoop
        server       = "hpa.sas.com"
        user         = XXXXX
        password     = YYYYY
        database     = myDB
        config       = "demo.xml" ;
```

For more information about LIBNAME options available for the Hadoop engine, see the LIBNAME topic in the Hadoop section of *SAS/ACCESS for Relational Databases: Reference*. The configuration file that you specify in the CONFIG= option contains information that is needed to access the Hive server. It also contains information that enables this configuration file to be used to access data in HDFS without using the Hive server. This information can also be used to specify replication factors and block sizes that are used when the engine writes data to HDFS.

The following DATA step uses the Hadoop engine to distribute to HDFS the simData data set that was used in the previous sections. The engine creates metadata for the data set in Hive.

```
data hdoopLib.simData;
    set simData;
run;
```

After you have loaded data or if you are accessing preexisting data in HDFS that have metadata in Hive, you can access this data alongside HDFS by using high-performance analytical procedures. The following HPLOGISTIC procedure statements perform the analysis in alongside-HDFS mode. These statements are similar to the PROC HPLOGISTIC example in the previous sections.

```
proc hplogistic data=hdoopLib.simData;
   class a b c;
   model y = a b c x1 x2 x3;
   performance host = "compute_appliance.sas.com";
run;
```

Figure 3.14 shows the "Performance Information" and "Data Access Information" tables. You see that the procedure ran in distributed mode and that the input data were read in parallel asymmetric mode. The numeric results shown in Figure 3.15 agree with the previous analyses.

Figure 3.14 Alongside-HDFS Execution by Using the Hadoop Engine

The HPLOGISTIC Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	142
Number of Threads per Node	32

Figure 3.14 *continued*

Data Access Information			
Data	**Engine**	**Role**	**Path**
GRIDLIB.SIMDATA	HADOOP	Input	Parallel, Asymmetric

Figure 3.15 Alongside-HDFS Execution by Using the Hadoop Engine

Model Information	
Data Source	GRIDLIB.SIMDATA
Response Variable	y
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Parameter Estimates					
Parameter	**Estimate**	**Standard Error**	**DF**	**t Value**	**Pr > |t|**
Intercept	5.7011	0.2539	Infty	22.45	<.0001
a 0	-0.01020	0.06627	Infty	-0.15	0.8777
a 1	0	.	.	.	.
b 0	0.7124	0.06558	Infty	10.86	<.0001
b 1	0	.	.	.	.
c 0	0.8036	0.06456	Infty	12.45	<.0001
c 1	0	.	.	.	.
x1	0.01975	0.000614	Infty	32.15	<.0001
x2	-0.04728	0.003098	Infty	-15.26	<.0001
x3	-0.1017	0.009470	Infty	-10.74	<.0001

The Hadoop engine also enables you to access tables in HDFS that are stored in various formats and that are not registered in Hive. You can use the HDMD procedure to generate metadata for tables that are stored in the following file formats:

- delimited text
- fixed-record length binary
- sequence files
- XML text

To read any other kind of file in Hadoop, you can write a custom file reader plug-in in Java for use with PROC HDMD. For more information about LIBNAME options available for the Hadoop engine, see the LIBNAME topic in the Hadoop section of *SAS/ACCESS for Relational Databases: Reference*.

The following example shows how you can use PROC HDMD to register metadata for CSV data independently from Hive and then analyze these data by using high-performance analytical procedures. The CSV data in the

table csvExample.csv is stored in HDFS in the directory /user/demo/data. Each record in this table consists of the following fields, in the order shown and separated by commas.

1. a string of at most six characters
2. a numeric field with values of 0 or 1
3. a numeric field with real numbers

Suppose you want to fit a logistic regression model to these data, where the second field represents a target variable named Success, the third field represents a regressor named Dose, and the first field represents a classification variable named Group.

The first step is to use PROC HDMD to create metadata that are needed to interpret the table, as in the following statements:

```
libname hdoopLib hadoop
                 server       = "hpa.sas.com"
                 user         = XXXXX
                 password     = YYYYY
                 HDFS_PERMDIR = "/user/demo/data"
                 HDFS_METADIR = "/user/demo/meta"
                 config       = "demo.xml"
                 DBCREATE_TABLE_EXTERNAL=YES;

proc hdmd name=hdoopLib.csvExample data_file='csvExample.csv'
          format=delimited encoding=utf8 sep = ',';

     column Group    char(6);
     column Success  double;
     column Dose     double;
run;
```

The metadata that are created by PROC HDMD for this table are stored in the directory /user/demo/meta that you specified in the HDFS_METADIR = option in the preceding LIBNAME statement. After you create the metadata, you can execute high-performance analytical procedures with these data by using the hdoopLib libref. For example, the following statements fit a logistic regression model to the CSV data that are stored in csvExample.csv table.

```
proc hplogistic data=hdoopLib.csvExample;
    class Group;
    model Success = Dose;
    performance host     = "compute_appliance.sas.com"
                gridmode = asym;
run;
```

Figure 3.16 shows the results of this analysis. You see that the procedure ran in distributed mode and that the input data were read in parallel asymmetric mode. The metadata that you created by using the HDMD procedure have been used successfully in executing this analysis.

Figure 3.16 Alongside-HDFS Execution with CSV Data

The HPLOGISTIC Procedure

Performance Information	
Host Node	compute_appliance.sas.com
Execution Mode	Distributed
Number of Compute Nodes	142
Number of Threads per Node	32

Data Access Information			
Data	**Engine**	**Role**	**Path**
GRIDLIB.CSVEXAMPLE	HADOOP	Input	Parallel, Asymmetric

Model Information	
Data Source	GRIDLIB.CSVEXAMPLE
Response Variable	Success
Class Parameterization	GLM
Distribution	Binary
Link Function	Logit
Optimization Technique	Newton-Raphson with Ridging

Class Level Information		
Class	**Levels**	**Values**
Group	3	group1 group2 group3

Number of Observations Read	1000
Number of Observations Used	1000

Parameter Estimates

Parameter	Estimate	Standard Error	DF	t Value	Pr > |t|
Intercept	0.1243	0.1295	Infty	0.96	0.3371
Dose	-0.2674	0.2216	Infty	-1.21	0.2277

Output Data Sets

In the alongside-the-database mode, the data are read in distributed form, minimizing data movement for best performance. Similarly, when you write output data sets and a high-performance analytical procedure executes in distributed mode, the data can be written in parallel into the database.

For example, in the following statements, the HPLOGISTIC procedure executes in distributed mode by using eight nodes on the appliance to perform the logistic regression on work.simData:

```
proc hplogistic data=simData;
   class a b c;
   model y = a b c x1 x2 x3;
   id a;
```

```
   output out=applianc.simData_out pred=p;
   performance host="hpa.sas.com" nodes=8;
run;
```

The output data set applianc.simData_out is written in parallel into the database. Although the data are fed on eight nodes, the database might distribute the data on more nodes.

When a high-performance analytical procedure executes in single-machine mode, all output objects are created on the client. If the libref of the output data sets points to the appliance, the data are transferred to the database on the appliance. This can lead to considerable performance degradation compared to execution in distributed mode.

Many procedures in SAS software add the variables from the input data set when an observationwise output data set is created. The assumption of high-performance analytical procedures is that the input data sets can be large and contain many variables. For performance reasons, the output data set contains the following:

- variables that are explicitly created by the statement
- variables that are listed in the ID statement, as described in Chapter 4, "Shared Statistical Concepts" (*SAS/STAT User's Guide: High-Performance Procedures*)
- distribution keys or hash keys that are transferred from the input data set

Including this information enables you to add to the output data set information necessary for subsequent SQL joins without copying the entire input data set to the output data set.

Working with Formats

You can use SAS formats and user-defined formats with high-performance analytical procedures as you can with other procedures in the SAS System. However, because the analytic work is carried out in a distributed environment and might depend on the formatted values of variables, some special handling can improve the efficiency of work with formats.

High-performance analytical procedures examine the variables that are used in an analysis for association with user-defined formats. Any user-defined formats that are found by a procedure are transmitted automatically to the appliance. If you are running multiple high-performance analytical procedures in a SAS session and the analysis variables depend on user-defined formats, you can preprocess the formats. This step involves generating an XML stream (a file) of the formats and passing the stream to the high-performance analytical procedures.

Suppose that the following formats are defined in your SAS program:

```
proc format;
     value YesNo        1='Yes'        0='No';
     value checkThis    1='ThisisOne'  2='ThisisTwo';
     value $cityChar    1='Portage'    2='Kinston';
run;
```

The next group of SAS statements create the XML stream for the formats in the file *Myfmt.xml*, associate that file with the file reference myxml, and pass the file reference with the FMTLIBXML= option in the PROC HPLOGISTIC statement:

```
filename myxml 'Myfmt.xml';
libname  myxml XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;
proc format cntlout=myxml.allfmts;
run;


proc hplogistic data=six fmtlibxml=myxml;
   class wheeze cit age;
   format wheeze best4. cit $cityChar.;
   model wheeze = cit age;
run;
```

Generation and destruction of the stream can be wrapped in convenience macros:

```
%macro Make_XMLStream(name=tempxml);
    filename &name 'fmt.xml';
    libname  &name XML92 xmltype=sasfmt tagset=tagsets.XMLsuv;
    proc format cntlout=&name..allfmts;
    run;
%mend;

%macro Delete_XMLStream(fref);
    %let rc=%sysfunc(fdelete(&fref));
%mend;
```

If you do not pass an XML stream to a high-performance analytical procedure that supports the FMTLIBXML= option, the procedure generates an XML stream as needed when it is invoked.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of a high-performance analytical procedure.

You can also use the PERFORMANCE statement to control whether a high-performance analytical procedure executes in single-machine or distributed mode.

You can specify the following *performance-options* in the PERFORMANCE statement:

COMMIT=*n*

requests that the high-performance analytical procedure write periodic updates to the SAS log when observations are sent from the client to the appliance for distributed processing.

High-performance analytical procedures do not have to use input data that are stored on the appliance. You can perform distributed computations regardless of the origin or format of the input data, provided

that the data are in a format that can be read by the SAS System (for example, because a SAS/ACCESS engine is available).

In the following example, the HPREG procedure performs LASSO variable selection where the input data set is stored on the client:

```
proc hpreg data=work.one;
   model y = x1-x500;
   selection method=lasso;
   performance nodes=10 host='mydca' commit=10000;
run;
```

In order to perform the work as requested using 10 nodes on the appliance, the data set Work.One needs to be distributed to the appliance.

High-performance analytical procedures send the data in blocks to the appliance. Whenever the number of observations sent exceeds an integer multiple of the COMMIT= size, a SAS log message is produced. The message indicates the actual number of observations distributed, and not an integer multiple of the COMMIT= size.

DETAILS
requests a table that shows a timing breakdown of the procedure steps.

GRIDHOST=*"name"*

HOST=*"name"*
specifies the name of the appliance host in single or double quotation marks. If this option is specified, it overrides the value of the GRIDHOST environment variable.

GRIDMODE=SYM | ASYM

MODE=SYM | ASYM
is a deprecated option that specifies whether to run the high-performance analytical procedure in symmetric (SYM) mode or asymmetric (ASYM) mode. This option overrides the GRIDMODE environment variable.

The only time you need to specify this option is when you want to run in symmetric mode alongside a Teradata database. In all other cases, this option is not needed and the value that you specify is ignored.

GRIDTIMEOUT=*s*

TIMEOUT=*s*
specifies the time-out in seconds for a high-performance analytical procedure to wait for a connection to the appliance and establish a connection back to the client. The default is 120 seconds. If jobs are submitted to the appliance through workload management tools that might suspend access to the appliance for a longer period, you might want to increase the time-out value.

INSTALL=*"name"*

INSTALLLOC=*"name"*
specifies the directory in which the shared libraries for the high-performance analytical procedure are installed on the appliance. Specifying the INSTALL= option overrides the GRIDINSTALLLOC environment variable.

LASRSERVER=*"path"*

LASR=*"path"*

specifies the fully qualified path to the description file of a SAS LASR Analytic Server instance. If the input data set is held in memory by this LASR Analytic Server instance, then the procedure runs alongside LASR. This option is not needed to run alongside LASR if the DATA= specification of the input data uses a libref that is associated with a LASR Analytic Server instance. For more information, see the section "Alongside-LASR Distributed Execution" on page 20 and the *SAS LASR Analytic Server: Reference Guide*.

NODES=ALL | *n*

NNODES=ALL | *n*

specifies the number of nodes in the distributed computing environment, provided that the data are not processed alongside the database.

Specifying NODES=0 indicates that you want to process the data in single-machine mode on the client machine. If the input data are not alongside the database, this is the default. The high-performance analytical procedures then perform the analysis on the client. For example, the following sets of statements are equivalent:

```
proc hplogistic data=one;
   model y = x;
run;
```

```
proc hplogistic data=one;
   model y = x;
   performance nodes=0;
run;
```

If the data are not read alongside the database, the NODES= option specifies the number of nodes on the appliance that are involved in the analysis. For example, the following statements perform the analysis in distributed mode by using 10 units of work on the appliance that is identified in the HOST= option:

```
proc hplogistic data=one;
   model y = x;
   performance nodes=10 host="hpa.sas.com";
run;
```

If the number of nodes can be modified by the application, you can specify a NODES=*n* option, where *n* exceeds the number of physical nodes on the appliance. The SAS High-Performance Server Distributed Mode software then *oversubscribes* the nodes and associates nodes with multiple units of work. For example, on a system that has 16 appliance nodes, the following statements oversubscribe the system by a factor of 3:

```
proc hplogistic data=one;
   model y = x;
   performance nodes=48 host="hpa.sas.com";
run;
```

Usually, it is not advisable to oversubscribe the system because the analytic code is optimized for a certain level of multithreading on the nodes that depends on the CPU count. You can specify NODES=ALL if you want to use all available nodes on the appliance without oversubscribing the system.

If the data are read alongside the distributed database on the appliance, specifying a nonzero value for the NODES= option has no effect. The number of units of work in the distributed computing environment is then determined by the distribution of the data and cannot be altered. For example, if you are running alongside an appliance with 24 nodes, the NODES= option in the following statements is ignored:

```
libname GPLib greenplm server=gpdca user=XXX password=YYY
              database=ZZZ;
proc hplogistic data=gplib.one;
   model y = x;
   performance nodes=10 host="hpa.sas.com";
run;
```

NTHREADS=*n*

THREADS=*n*

specifies the number of threads for analytic computations and overrides the SAS system option THREADS | NOTHREADS. If you do not specify the NTHREADS= option, the number of threads is determined based on the number of CPUs on the host on which the analytic computations execute. The algorithm by which a CPU count is converted to a thread count is specific to the high-performance analytical procedure. Most procedures create one thread per CPU for the analytic computations.

By default, high-performance analytical procedures run in multiple concurrent threads unless multithreading has been turned off by the NOTHREADS system option or you force single-threaded execution by specifying NTHREADS=1. The largest number that can be specified for *n* is 256. Individual high-performance analytical procedures can impose more stringent limits if called for by algorithmic considerations.

NOTE: The SAS system options THREADS | NOTHREADS apply to the client machine on which the SAS high-performance analytical procedures execute. They do not apply to the compute nodes in a distributed environment.

Chapter 4
The HPBIN Procedure

Contents

Overview: HPBIN Procedure

Binning is a common step in the data preparation stage of the model-building process. You can use binning to classify missing variables, reduce the impact of outliers, and generate multiple effects. The generated effects are useful and contain certain nonlinear information about the original interval variables.

The HPBIN procedure conducts high-performance binning by using binning methods that are described in the following subsections. The HPBIN procedure can also calculate the weight of evidence (WOE) and information value (IV) based on binning results.

The HPBIN procedure runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Server Distributed Mode.

Bucket Binning

Bucket binning creates equal-length bins and assigns the data to one of these bins. You can choose the number of bins during the binning; the default number of bins (the binning level) is 16.

Winsorized Binning

Winsorized binning is similar to bucket binning except that both tails are cut off to obtain a smooth binning result. This technique is often used to remove outliers during the data preparation stage.

Quantile Binning

Quantile binning aims to assign the same number of observations to each bin, if the number of observations is evenly divisible by the number of bins. As a result, each bin should have the same number of observations, provided that there are no tied values at the boundaries of the bins. Because PROC HPBIN always assigns observations that have the same value to the same bin, quantile binning might create unbalanced bins if any variable has tied values. For more information, see the section "Binning Computation and Formulas" on page 51.

Pseudo–Quantile Binning

The HPBIN procedure offers pseudo–quantile binning, which is an approximation of quantile binning. The pseudo–quantile binning method is very efficient, and the results mimic those of the quantile binning method.

PROC HPBIN Features

The HPBIN procedure has the following features:

- provides a bucket (equal-length) binning method
- provides a Winsorized binning method and Winsorized statistics
- provides a quantile binning method and a pseudo–quantile binning method
- provides a mapping table for the selected binning method
- provides a basic statistical table that contains the minimum, maximum, mean, median, and so on
- computes the quantiles of the binning variables
- calculates the weight of evidence (WOE) and information value (IV) based on binning results
- reads input data in parallel and writes output data in parallel when the data source is in a database on the appliance

Because the HPBIN procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Getting Started: HPBIN Procedure

This example shows how you can use the HPBIN procedure to perform pseudo–quantile binning. Consider the following statements:

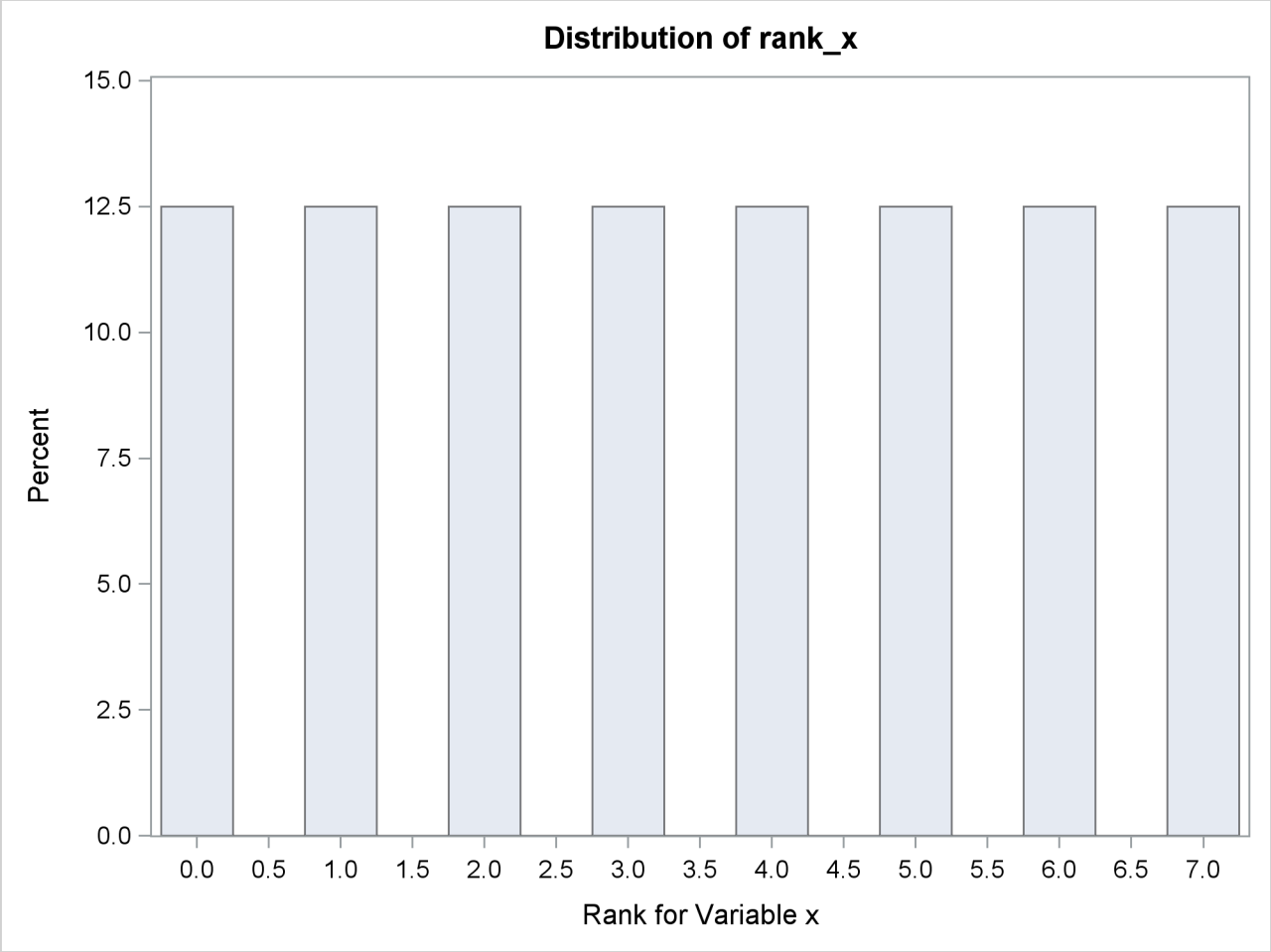
```
    data bindata;
        do i=1 to 1000;
         x=rannorm(1);
         output;
        end;
    run;

proc rank data=bindata out=rankout group=8;
    var x;
    ranks rank_x;
run;

ods graphics on;
proc univariate data=rankout;
    var rank_x;
    histogram;
run;
```

These statements create a data set that contains 1,000 observations, each generated from a random normal distribution. The histogram for this data set is shown in Figure 4.1.

Figure 4.1 Histogram for Rank X



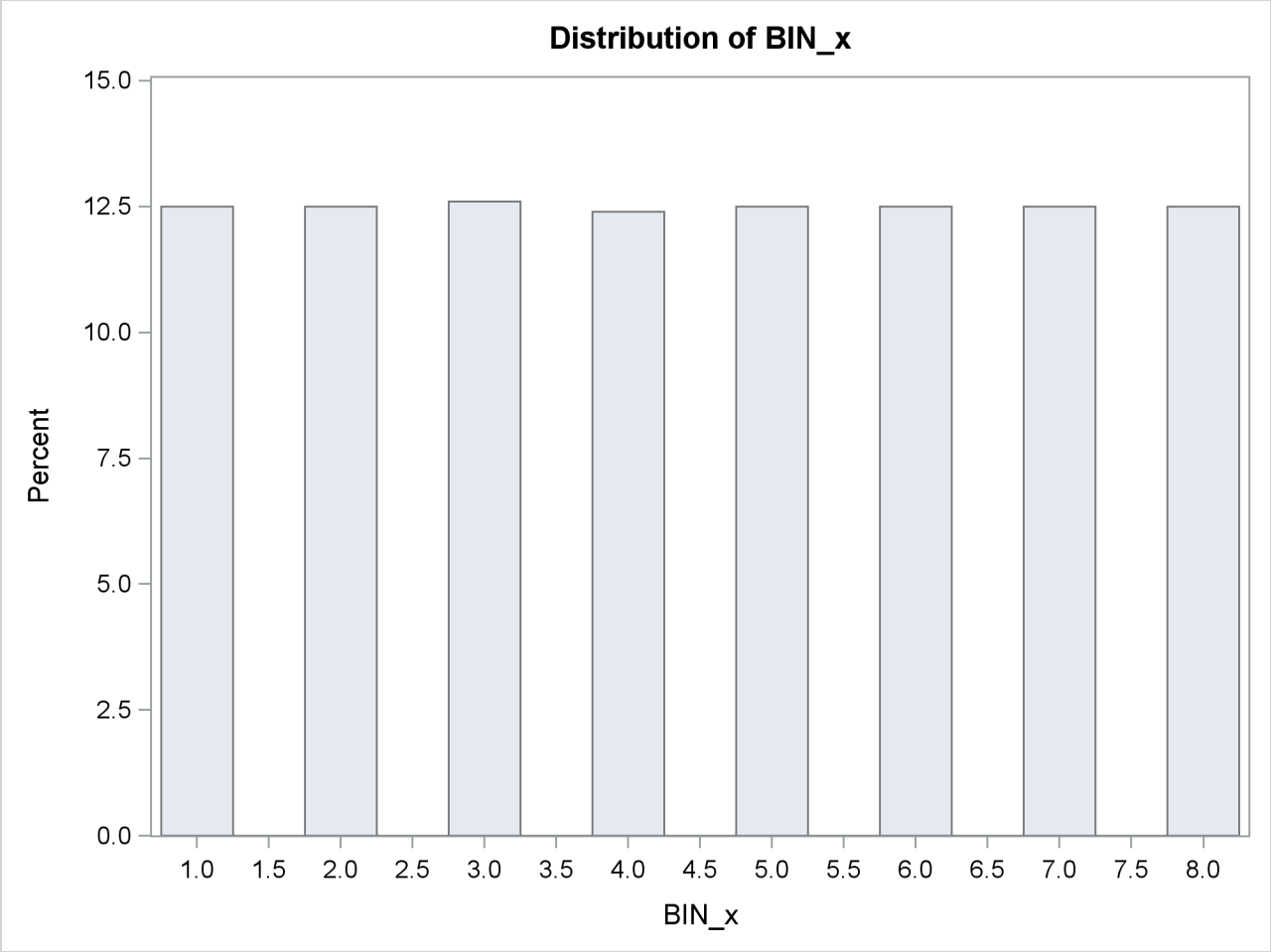
The pseudo–quantile binning method in the HPBIN procedure can achieve a similar result by using far less computation time. In this case, the time complexity is $C * O(n)$, where C is a constant and n is the number of observations. When the algorithm runs on the grid, the total amount of computation time is much less. For example, if a cluster has 32 nodes and each node has 24 shared-memory CPUs, then the time complexity is $(C * O(n))/(32 * 24)$.

The following statements bin the data by using the PSEUDO_QUANTILE option in the PROC HPBIN statement and generate the histogram for the binning output data. (See Figure 4.2.) This histogram is similar to the one in Figure 4.1.

```
proc hpbin data=bindata output=binout numbin=8 pseudo_quantile;
    input x;
run;

ods graphics on;
proc univariate data=binout;
    var bin_x;
    histogram;
run;
```

Figure 4.2 Histogram for Binning X



Syntax: HPBIN Procedure

The following statements are available in the HPBIN procedure:

PROC HPBIN < *options* > **;**
CODE FILE=*filename* **;**
FREQ *variable* **;**
ID *variables* **;**
INPUT *variables* < / *option* > **;**
PERFORMANCE < *performance-options* > **;**
TARGET *variable* / **LEVEL**=*level* **ORDER**=*order* **;**

The PROC HPBIN statement is required. You can specify multiple INPUT statements. The following sections describe the PROC HPBIN statement and then describe the other statements in alphabetical order.

PROC HPBIN Statement

PROC HPBIN < *options* > **;**

The PROC HPBIN statement invokes the procedure. Table 4.1 summarizes important *options* in the PROC HPBIN statement by function.

Table 4.1 PROC HPBIN Statement Options

Option	Description
Basic Options	
DATA=	Specifies the input data set
OUTPUT=	Specifies the output data set
NOPRINT	Overwrites the ODS output
Binning Level Options	
NUMBIN=	Specifies the global number of bins for all binning variables
Binning Method Options	
BUCKET	Specifies the bucket binning method
PSEUDO_QUANTILE	Specifies the pseudo–quantile binning method
QUANTILE	Specifies the quantile binning method
WINSOR	Specifies the Winsorized binning method and the rate that it uses
WINSORRATE=	
Statistics Options	
COMPUTESTATS	Computes the basic statistics of the binning variables
COMPUTEQUANTILE	Compute the quantiles of the binning variables
Weight-of-Evidence Options	
BINS_META=	Specifies the BINS_META input data set, which contains the binning results
WOE	Computes the weight of evidence and information values
WOEADJUST=	Specifies the adjustment factor for the weight-of-evidence calculation

You can specify the following optional arguments:

BINS_META=*SAS-data-set*
specifies the BINS_META input data set, which contains the binning results. The BINS_META data set contains six variables: variable name, binned variable name, lower bound, upper bound, bin, and range. The mapping table that is generated by PROC HPBIN can be used as the BINS_META data set.

BUCKET | QUANTILE | PSEUDO_QUANTILE | WINSOR WINSORRATE=*number*
specifies which binning method to use. If you specify BUCKET, then PROC HPBIN uses equal-length binning. If you specify QUANTILE, then PROC HPBIN uses quantile binning. If you specify PSEUDO_QUANTILE, then PROC HPBIN generates a result that approximates the quantile binning. If you specify WINSOR, PROC HPBIN uses Winsorized binning, and you must specify the

WINSORRATE option with a value from 0.0 to 0.5 exclusive for *number*. You can specify only one option. The default is BUCKET.

However, when a BINS_META data set is specified, PROC HPBIN does not do binning and ignores the binning method options, binning level options, and INPUT statement. Instead, PROC HPBIN takes the binning results from the BINS_META data set and calculates the weight of evidence and information value.

COMPUTEQUANTILE
computes the quantile result. If you specify COMPUTEQUANTILE, PROC HPBIN generates the quantiles and extremes table, which contains the following percentages: 0% (Min), 1%, 5%, 10%, 25% (Q1), 50% (Median), 75% (Q3), 90%, 95%, 99%, and 100% (Max).

COMPUTESTATS
computes the statistic result. If you specify COMPUTESTATS, basic statistical information is computed and ODS output can be provided. The output table contains six variables: the mean, median, standard deviation, minimum, maximum, and number of bins for each binning variable.

DATA=*SAS-data-set*
specifies the input SAS data set or database table to be used by PROC HPBIN.

If the procedure executes in distributed mode, the input data are distributed to memory on the appliance nodes and analyzed in parallel, unless the data are already distributed in the appliance database. In this case, PROC HPBIN reads the data alongside the distributed database.

For single-machine mode, the input must be a SAS data set.

NOPRINT
suppresses the generation of ODS outputs.

NUMBIN=*integer*
specifies the global number of binning levels for all binning variables. The value of *integer* can be any integer between 2 and 1,000, inclusive. The default number of binning levels is 16.

The resulting number of binning levels might be less than the specified *integer* if the sample size is small or if the data are not normalized. In this case, PROC HPBIN provides a warning message.

You can specify a different number of binning levels for each different variable in an INPUT statement. The number of binning levels that you specify in an INPUT statement overwrites the global number of binning levels.

OUTPUT=*SAS-data-set*
creates an output SAS data set in single-machine mode or a database table that is saved alongside the distributed database in distributed mode. The output data set or table contains binning variables. To avoid data duplication for large data sets, the variables in the input data set are not included in the output data set.

WOE
computes the weight of evidence (WOE) and information value (IV).

WOEADJUST=*number*

specifies the adjustment factor for the weight-of-evidence calculation. You can specify any value from 0.0 to 1.0, inclusive, for *number*. The default is 0.5.

CODE Statement

CODE FILE=*filename* **;**

The CODE statement is optional in PROC HPBIN. If you use a CODE statement, score code is generated and stored in a file that can be used for scoring purposes. Only one CODE statement is processed. If you specify multiple CODE statements, only the first one is used.

FREQ Statement

FREQ *variable* **;**

The *variable* in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. The HPBIN procedure treats each observation as if it appeared *n* times, where *n* is the value of the *variable* for the observation. If *n* is not an integer, it is truncated to an integer. If *n* is less than 1 or is missing, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

ID Statement

ID *variables* **;**

The optional ID statement lists one or more variables from the input data set to be transferred to the output data set. The ID statement accepts both numeric and character variables. The variables in an ID statement can also appear in an INPUT statement.

To avoid data duplication for large data sets, the HPBIN procedure does not include any variables from the input data set in the output data set by default. Therefore, the ID statement can be used to copy variables from the input data set to the output data set.

INPUT Statement

INPUT *variables* < / *option* > **;**

The INPUT statement names one or more *variables* as input variables for binning. The specified *variables* must be interval variables. If classification variables are provided, PROC HPBIN stops with an error message. PROC HPBIN does not support duplicate variables. If the INPUT statement contains a duplicate variable, PROC HPBIN takes only the first variable and provides a warning message. You can specify the following *option* in each INPUT statement:

NUMBIN=*integer*

specifies the number of binning levels for all the binning variables in the current INPUT statement. The value of *integer* can be any integer between 2 and 1,000, inclusive.

The resulting number of binning levels might be less than the specified *integer* if the sample size is small or if the data are not normalized. In these cases, PROC HPBIN provides a warning message.

The number of binning levels that you specify in an INPUT statement overwrites the global number of binning levels. If you do not specify the NUMBIN= option in the INPUT statement, PROC HPBIN uses the global number of binning levels, which is 16 by default but can be specified in the NUMBIN= option in the PROC HPBIN statement.

When a BINS_META data set is specified, PROC HPBIN does not do binning and ignores the INPUT statement. Instead, PROC HPBIN takes the binning results from the BINS_META data set and calculates the weight of evidence and information value for the interval variables in the BINS_META data set.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of the HPBIN procedure.

You can also use the PERFORMANCE statement to control whether the HPBIN procedure executes in single-machine or distributed mode.

The PERFORMANCE statement is documented further in the section "PERFORMANCE Statement" on page 36 of Chapter 3, "Shared Concepts and Topics."

TARGET Statement

TARGET *variable* / **LEVEL=***level* **ORDER=***order* **;**

The TARGET statement names the *variable* that PROC HPBIN uses to calculate the weight of evidence and information value. You can specify the following arguments:

LEVEL=*level*

specifies the level of the target variable. The values of *level* can be BINARY or NOMINAL. The default is NOMINAL.

ORDER=*order*

specifies the order of the target variable. The values of *order* can be ASCENDING or DESCENDING. The default is DESCENDING.

Details: HPBIN Procedure

Computing the Quantiles (Percentiles)

PROC HPBIN computes the 0% (minimum), 1%, 5%, 10%, 25%, 50%, 75%, 90%, 95%, 99%, and 100% (maximum) percentiles of each binning variable.

Let m be the number of nonmissing values for a variable, and let $x_1, x_2, \ldots, x_m$ represent the ordered values of the variable. Let the tth percentile be y, set $p = \frac{t}{100}$, and let $mp = j + g$, where j is the integer part of mp and g is the fractional part of mp. Then the tth percentile is described as:

$$y = \begin{cases} x_j & \text{if } g = 0 \\ x_{j+1} & \text{if } g > 0 \end{cases}$$

NOTE: To avoid the time-consuming sorting process, the HPBIN procedure uses an iterative projection method to compute percentiles according to the preceding formula.

Binning Computation and Formulas

For variable x, assume that the data set is $\{x_i\}$, where $i = 1, 2, \ldots, n$. Let $\min(x) = \min_{i \in \{1..n\}} \{x_i\}$, and let $\max(x) = \max_{i \in \{1..n\}} \{x_i\}$. The range of the variable is $\text{range}(x) = \max(x) - \min(x)$.

The computations for the various binning methods are as follows:

- For bucket binning, the length of the bucket is

  $$L = \frac{\max(x) - \min(x)}{n}$$

  The split points are

  $$s_k = \min(x) + L * k$$

  where $k = 1, 2, \ldots,$ *numbin* $- 1$, and *numbin* is the value of the NUMBIN= option in the PROC HPBIN statement.

  When the data are evenly distributed on the SAS appliance, the time complexity for bucket binning is $\mathcal{O}(n)/(nodes * cpus)$, where n is the number of observations, *nodes* is the number of computer nodes on the appliance, and *cpus* is the number of CPUs on each node.

- For quantile binning, PROC HPBIN calculates a quantile table P. Let $P = \{p_k\}$, where $k = 1, 2, \ldots,$ *numbin*. Then p_k is described as

  $$p_k = \begin{cases} 1.0/numbin & \text{if } k = 0 \\ 1.0/numbin + p_{k-1} & \text{if } 0 < k < numbin \\ 1.0 & \text{if } k = numbin \end{cases}$$

  Quantile binning often requires data to be sorted in a particular way, and the sorting process usually consumes a significant amount of CPU time and memory. When the input data set is larger than the

available memory, the sorting algorithm becomes more complicated. In distributed computing, data communications overhead also increases the sorting challenge. To avoid the time-consuming sorting process, the HPBIN procedure uses an iterative projection method for quantile binning, which runs much faster than sorting-based quantile binning method in most cases.

After calculating the quantile table, PROC HPBIN uses an iterative projection method to compute quantiles (percentiles) according to the formula that is described in the section "Computing the Quantiles (Percentiles)" on page 51.

Quantile binning aims to assign the same number of observations to each bin, if the number of observations is evenly divisible by the number of bins. As a result, each bin should have the same number of observations, provided that there are no tied values at the boundaries of the bins. Because PROC HPBIN always assigns observations that have the same value to the same bin, quantile binning might create unbalanced bins if any variable has tied values. For example, if an observation whose value is x is assigned to bin k, then every observation whose value is x is assigned to bin k for this variable, and no observation whose value is x is assigned to the next bin, bin $k + 1$. Therefore, bin k might have more observations than bin $k + 1$, because the tied values at the boundaries between bin k and bin $k + 1$ are all assigned to bin k. That is, tied values at the boundaries between two bins are always assigned to the lower-numbered bin.

- For pseudo–quantile binning and Winsorized binning, the sorting algorithm is more complex to describe but very efficient to execute. For variable x, PROC HPBIN uses a simple bucket sorting method to obtain the basic information. Let N = 10,000 be the number of buckets, ranging from $\min(x)$ to $\max(x)$. For each bucket B_i, $i = 1, 2, \ldots, N$, PROC HPBIN keeps following information:

  - c_i: count of x in B_i
  - $\min_i$: minimum value of x in B_i
  - $\max_i$: maximum value of x in B_i
  - $\sum_i$: sum of x in B_i
  - $\sum 2_i$: sum of x^2 in B_i

  To calculate the quantile table, let $P = \{0.00, 0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.99, 1.00\}$. For each $p_k \in P$, $k = 1, 2, \ldots, 11$, find the smallest I_k such that $\sum_{i=1}^{I_k} c_i \geq p_k * n$. Therefore, the quantile value Q_k is obtained,

$$Q_k = \begin{cases} min_{I_k} & \text{if } \sum_{i=1}^{I_k} c_i > p_k * n \\ max_{I_k} & \text{if } \sum_{i=1}^{I_k} c_i = p_k * n \end{cases}$$

  where $k = 1, 2, \ldots, 11$.

PROC HPBIN calculates the split points as follows:

- For pseudo–quantile binning, let the base count $bc = \text{ceil}(\frac{n}{numbin})$. PROC HPBIN finds those integers $\{I_k | k = 1, 2, \dots\}$ such that

$$\left(\sum_{i=1}^{I_k} c_i \geq \sum_{i=1}^{I_{k-1}} c_i + bc \quad \text{or} \quad \sum_{i=1}^{I_k} c_i \geq \frac{n}{numbin} * k \right)$$
$$\text{and} \quad \sum_{i=1}^{I_k} c_i > \sum_{i=1}^{I_{k-1}} c_i$$
$$\text{and} \quad \sum_{i=1}^{I_k} c_i < n$$

  where k is the kth split.

  The split value is

$$s_k = \min(x) + \frac{\max(x) - \min(x)}{N} * I_k$$

  where $k = 1, 2, \dots$, and $k <$ *numbin*.

  The time complexity for pseudo–quantile binning is $C\mathcal{O}(n)/(nodes * cpus)$, where C is a constant that depends on the number of sorting bucket N, n is the number of observations, *nodes* is the number of computer nodes on the appliance, and *cpus* is the number of CPUs on each node.

- For Winsorized binning, the Winsorized statistics are computed first. After the minimum and maximum have been found, the split points are calculated the same way as in bucket binning.

  Let the tail count wc be $\text{ceil}(\text{WinsorRate} * n)$, and find the smallest I such that $\sum_{i=1}^{I} c_i \geq \text{wc}$. Then, the left tail count is $\text{lwc} = \sum_{i=1}^{I} c_i$. Find the next I_l such that $\sum_{i=1}^{I_l} c_i > lwc$. Therefore, the minimum value is $\text{WinsorMin} = \min_{I_l}$. Similarly, find the largest I such that $\sum_{i=I}^{N} c_i \geq \text{wc}$. The right tail count is $\text{rwc} = \sum_{i=I}^{N} c_i$. Find the next I_r such that $\sum_{i=I_r}^{N} c_i > \text{rwc}$. Then the maximum value is $\text{WinsorMax} = \max_{I_r}$. The mean is calculated by the formula

$$\text{WinsorMean} = \frac{\text{lwc} * \text{WinsorMin} + \sum_{i=I_l}^{I_r} sum_i + \text{rwc} * \text{WinsorMax}}{n}$$

  The trimmed mean is calculated by the formula

$$\text{trimmedMean} = \frac{\sum_{i=I_l}^{I_r} sum_i}{n - \text{lwc} - \text{rwc}}$$

NOTE: PROC HPBIN prints an error or a warning message when the results might not be accurate.

NOTE: PROC HPBIN does not allow empty bins. If an empty bin is detected because of an insufficient number of nonmissing observations, PROC HPBIN issues an error and exits.

NOTE: If PROC HPBIN detects an empty bin followed by a bin that is not empty, it skips the empty bin and does not assign a number to it. In this case, the number of bins that PROC HPBIN generates is less than the specified number of bins.

Computing the Weight of Evidence and Information Value

PROC HPBIN can compute the weight of evidence and the information value.

Weight of evidence (WOE) is a measure of how much the evidence supports or undermines a hypothesis. WOE measures the relative risk of an attribute of binning level. The value depends on whether the value of the target variable is a non-event or an event. An attribute's WOE is defined as follows:

$$WOE_{attribute} = ln\frac{p^{non-event}_{attribute}}{p^{event}_{attribute}} = ln\frac{\frac{N^{attribute}_{non-event}}{N^{total}_{non-event}}}{\frac{N^{attribute}_{event}}{N^{total}_{event}}}$$

The definitions of the quantities in the preceding formula are as follows:

- $N^{attribute}_{non-event}$: the number of non-event records that exhibit the attribute
- $N^{total}_{non-event}$: the total number of non-event records
- $N^{attribute}_{event}$: the number of event records that exhibit the attribute
- N^{total}_{event}: the total number of event records

To avoid an undefined WOE, an adjustment factor, x, is used:

$$WOE_{attribute} = ln\frac{\frac{N^{attribute}_{non-event}+x}{N^{total}_{non-event}}}{\frac{N^{attribute}_{event}+x}{N^{total}_{event}}}$$

You can use the WOEADJUST= option to specify a value between [0, 1] for x. By default, x is 0.5.

The information value (IV) is a weighted sum of the WOE of the characteristic's attributes. The weight is the difference between the conditional probability of an attribute given an event and the conditional probability of that attribute given a non-event. In the following formula of IV, m is the number of bins of a variable:

$$IV = \sum_{i=1}^{m}(\frac{N^{attribute}_{non-event}}{N^{total}_{non-event}} - \frac{N^{attribute}_{event}}{N^{total}_{event}}) * WOE_i$$

An information value can be any real number. Generally speaking, the higher the information value, the more predictive a characteristic is likely to be.

Data Output

The HPBIN procedure can write an output table back to the database. If the input data are from the appliances and the processing is alongside the database, PROC HPBIN writes the output table back to each appliance in parallel. For single-machine mode, the output table is a SAS data set. In this case, you can still take advantage of multithreading, which performs parallel computation on a single node.

If you do not specify the OUTPUT option in the PROC HPBIN statement, the write-back process is skipped. This is useful when only the basic statistical tables are computed.

By default, the output table contains the new binned variables. If you specify an ID statement, the output table contains the IDs and the binned variables.

NOTE: If an input variable value is missing, then the binning output level value is 0.

ODS Tables

Each table that the HPBIN procedure creates has a name associated with it. You must use this name to refer to the table in ODS statements. These table names are listed in Table 4.2.

Table 4.2 ODS Tables Produced by PROC HPBIN

Table Name	Description	Statement	Options
BinInfo	Basic binning information and parameters	PROC HPBIN	Default
InfoValue	Information value for each variable	PROC HPBIN	WOE
Mapping	Level mapping information	PROC HPBIN	Default
NObs	Number of observations read and used	PROC HPBIN	WOE
PerformanceInfo	Information about the high-performance computing environment	PROC HPBIN	Default
Quantile	Quantiles and extremes	PROC HPBIN	COMPUTEQUANTILE
Summary	Summary statistics for the given variables	PROC HPBIN	COMPUTESTATS
Trim	Trimmed statistics for the given variables	PROC HPBIN	WINSOR, WINSORRATE
Winsor	Winsor statistics for the given variables	PROC HPBIN	WINSOR, WINSORRATE
WOE	Weight of evidence for each bin	PROC HPBIN	WOE
Timing	Timing	PERFORMANCE	DETAILS

The following list provides more information about these tables:

BinInfo By default, PROC HPBIN generates the "Binning Information" table in its ODS output. This table displays some procedure parameters and data information, which includes the binning method, the number of bins, and the number of variables.

InfoValue PROC HPBIN generates the "Information Value" table if you specify the WOE option in the PROC HPBIN statement. This table provides the information value for each variable.

Mapping By default, PROC HPBIN generates a "Mapping" table in its ODS output. This table provides the level mapping for the input variables. The level starts at 1 and increases to the value that you specify in the NUMBIN= option. In the mapping table, a missing value for the lower bound indicates negative infinity, and a missing value for the upper bound indicates positive infinity.

The final bin level can be less than the NUMBIN value if the input data are small or the binning variable is discrete. In this case, a warning message is printed in the log.

NObs PROC HPBIN generates the "Nobs" table if you specify the WOE option. This table provides the number of observations that are read and used.

PerformanceInfo By default, PROC HPBIN produces the "Performance Information" table. It displays information about the execution mode. For single-machine mode, the table displays the number of threads used. For distributed mode, the table displays the grid mode (symmetric or asymmetric), the number of compute nodes, and the number of threads per node.

Quantile PROC HPBIN generates the "Quantiles and Extremes" table if you specify the COMPUTEQUANTILE option. This table contains the following quantile levels for each variable: 0% (Min), 1%, 5%, 10%, 15%, 25% (Q1), 50% (Median), 75% (Q3), 90%, 95%, 99%, and 100% (Max).

Summary PROC HPBIN generates the "Summary Statistics" table if you specify the COMPUTESTATS option. This table displays the variable name, number of nonmissing observations, number of missing observations, mean, median, standard deviation, minimum, maximum, and number of bins.

Trim PROC HPBIN also generates the "Trimmed Statistics" table if you specify the WINSOR option. This table contains the trimmed minimum, maximum, mean, standard error mean, left tail, left tail percentage, right tail, and right tail percentage, and the degrees of freedom (DF).

Winsor PROC HPBIN generates the "Winsor Statistics" table if you specify the WINSOR option. The "Winsor Statistics" table contains the Winsorized minimum, maximum, mean, standard error mean, left tail, left tail percentage, right tail, and right tail percentage, and the degrees of freedom (DF).

WOE PROC HPBIN generates the "Weight of Evidence" table if you specify the WOE option in the PROC HPBIN statement. This table provides the level mapping information, binning information, weight of evidence, and information value for each bin.

When the "Weight of Evidence" table is printed, the "Mapping" table is not printed because the level mapping information is the same in both tables.

In addition to the level mapping information, the "Weight of Evidence" table contains some other information such as the non-event count, non-event rate, event count, event rate, weight of evidence, and information value for each bin.

Examples

Example 4.1: Bucket Binning in Single-Machine Mode

This example shows how you can use PROC HPBIN in single-machine mode.

The following DATA step creates the SAS data set as the input for the example:

```
data ex12;
    length id 8;
    do id=1 to 1000;
        x1 = ranuni(101);
        x2 = 10*ranuni(201);
        x3 = 100*ranuni(301);
        output;
    end;
run;
```

The following statements show the basic usage:

```
proc hpbin data=ex12 output=out numbin=10 bucket ;
    input x1-x3;
run;

proc print data=out(obs=10); run;
```

PROC HPBIN generates several ODS tables, which are shown in Output 4.1.1 through Output 4.1.2.

Output 4.1.1 PROC HPBIN Binning Information Display

The HPBIN Procedure

Binning Information	
Method	Bucket Binning
Number of Bins Specified	10
Number of Variables	3

Output 4.1.2 PROC HPBIN Mapping Table

Mapping				
Variable	Binned Variable	Range	Frequency	Proportion
x1	**BIN_x1**	x1 < 0.099925	108	0.10800000
		0.099925 <= x1 < 0.199399	94	0.09400000
		0.199399 <= x1 < 0.298872	90	0.09000000
		0.298872 <= x1 < 0.398345	95	0.09500000
		0.398345 <= x1 < 0.497818	90	0.09000000
		0.497818 <= x1 < 0.597291	115	0.11500000
		0.597291 <= x1 < 0.696764	116	0.11600000
		0.696764 <= x1 < 0.796237	105	0.10500000
		0.796237 <= x1 < 0.895711	106	0.10600000
		0.895711 <= x1	81	0.08100000
x2	**BIN_x2**	x2 < 1.021539	99	0.09900000
		1.021539 <= x2 < 2.015407	110	0.11000000
		2.015407 <= x2 < 3.009275	99	0.09900000
		3.009275 <= x2 < 4.003143	85	0.08500000
		4.003143 <= x2 < 4.997011	109	0.10900000
		4.997011 <= x2 < 5.990878	98	0.09800000
		5.990878 <= x2 < 6.984746	100	0.10000000
		6.984746 <= x2 < 7.978614	94	0.09400000
		7.978614 <= x2 < 8.972482	112	0.11200000
		8.972482 <= x2	94	0.09400000
x3	**BIN_x3**	x3 < 10.173597	97	0.09700000
		10.173597 <= x3 < 20.142996	97	0.09700000
		20.142996 <= x3 < 30.112396	106	0.10600000
		30.112396 <= x3 < 40.081796	94	0.09400000
		40.081796 <= x3 < 50.051196	102	0.10200000
		50.051196 <= x3 < 60.020596	99	0.09900000
		60.020596 <= x3 < 69.989995	107	0.10700000
		69.989995 <= x3 < 79.959395	92	0.09200000
		79.959395 <= x3 < 89.928795	91	0.09100000
		89.928795 <= x3	115	0.11500000

In this example, PROC HPBIN also generates the output table. The first 10 observations of the table are shown in Output 4.1.3.

Output 4.1.3 First 10 Observations in the Output Table

Obs	BIN_x1	BIN_x2	BIN_x3
1	7	10	4
2	2	1	9
3	9	8	1
4	8	8	10
5	2	8	7
6	1	6	7
7	7	9	6
8	7	5	1
9	1	4	1
10	7	9	1

Example 4.2: Pseudo–Quantile Binning in Distributed Mode

This example shows pseudo–quantile binning that is executed in distributed mode. The following DATA step generates 1,000,000 observations:

```
data ex12;
    length id 8;
    do id=1 to 1000000;
        x1 = ranuni(101);
        x2 = 10*ranuni(201);
        output;
    end;
run;
```

You can run PROC HPBIN in distributed mode by specifying valid values for the NODES=, INSTALL=, and HOST= options in the PERFORMANCE statement. An alternative to specifying the INSTALL= and HOST= options in the PERFORMANCE statement is to set appropriate values for the GRIDHOST and GRIDINSTALLLOC environment variables by using OPTIONS SET commands. See the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics," for details about setting these options or environment variables.

The following statements provide an example. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to macro variables with appropriate values.

```
ods output BinInfo=bininfo;
ods output PerformanceInfo=perfInfo;
ods output Mapping=mapTable;
ods output Summary=Summary;
ods output Quantile=Quantile;
ods listing close;
proc hpbin data=ex12 output=out numbin=10 pseudo_quantile
           computestats computequantile ;
    input x1-x2;
    performance nodes=4 nthreads=8
    host="&GRIDHOST" install="&GRIDINSTALLLOC";
run;
ods listing;
```

The "Performance Information" table in Output 4.2.1 shows the grid setting.

Output 4.2.1 PROC HPBIN Performance Information

Performance Information	
Host Node	<< your grid host >>
Install Location	<< your grid install location >>
Execution Mode	Distributed
Grid Mode	Symmetric
Number of Compute Nodes	4
Number of Threads per Node	8

The "Binning Information" table in Output 4.2.2 shows the binning method, number of bins, and number of variables.

Output 4.2.2 PROC HPBIN Binning Information

Binning Information	
Method	Pseudo-Quantile Binning
Number of Bins Specified	10
Number of Variables	2

The "Mapping" table in Output 4.2.3 shows the level mapping of the input variables.

Output 4.2.3 PROC HPBIN Mapping

Mapping				
Variable	Binned Variable	Range	Frequency	Proportion
x1	**BIN_x1**	x1 < 0.099900	100046	0.10004600
		0.099900 <= x1 < 0.199500	100029	0.10002900
		0.199500 <= x1 < 0.299300	100016	0.10001600
		0.299300 <= x1 < 0.399500	99939	0.09993900
		0.399500 <= x1 < 0.500000	100049	0.10004900
		0.500000 <= x1 < 0.599800	99989	0.09998900
		0.599800 <= x1 < 0.700400	99975	0.09997500
		0.700400 <= x1 < 0.800300	100014	0.10001400
		0.800300 <= x1 < 0.900299	100007	0.10000700
		0.900299 <= x1	99936	0.09993600
x2	**BIN_x2**	x2 < 0.997008	100006	0.10000600
		0.997008 <= x2 < 1.995006	100025	0.10002500
		1.995006 <= x2 < 2.994005	99986	0.09998600
		2.994005 <= x2 < 3.995004	100034	0.10003400
		3.995004 <= x2 < 4.999002	99990	0.09999000
		4.999002 <= x2 < 5.998001	100063	0.10006300
		5.998001 <= x2 < 6.993000	99929	0.09992900
		6.993000 <= x2 < 7.998998	100008	0.10000800
		7.998998 <= x2 < 8.999997	100010	0.10001000
		8.999997 <= x2	99949	0.09994900

The "Summary Statistics" table in Output 4.2.4 displays the basic statistical information, including the number of observations, number of missing observations, mean, median, and so on.

Output 4.2.4 PROC HPBIN Summary Statistics Table

Summary Statistics								
Variable	N	N Missing	Mean	Median	Std Dev	Minimum	Maximum	N Bins
x1	1000000	0	0.49984213	0.49991238	0.28894736	2.24449E-7	0.99999939	10
x2	1000000	0	4.99688234	4.99851593	2.88736227	9.10833E-6	9.99999537	10

The "Quantiles and Extremes" table in Output 4.2.5 shows the quantile computation of the variables. The ODS table is generated only when the COMPUTESTATS option is specified in the PROC HPBIN statement.

Output 4.2.5 PROC HPBIN Quantile Computation

Quantiles and Extremes		
Variable	Quantile Level	Quantile
x1	Max	0.99999939
	.99	0.99011639
	.95	0.95024946
	.90	0.90023557
	.75 Q3	0.75032495
	.50 Median	0.49991238
	.25 Q1	0.24931534
	.10	0.09985729
	.05	0.04954403
	.01	0.01000524
	Min	2.24449E-7
x2	Max	9.99999537
	.99	9.90136979
	.95	9.49989152
	.90	8.99939011
	.75 Q3	7.49894200
	.50 Median	4.99851593
	.25 Q1	2.49431827
	.10	0.99691767
	.05	0.49879104
	.01	0.10062442
	Min	9.10833E-6

Example 4.3: Quantile Binning in Distributed Mode

This example shows quantile binning that is executed in distributed mode. Most of this example is the same as the pseudo–quantile binning example (see "Example 4.2: Pseudo–Quantile Binning in Distributed Mode"), so you can easily compare these two binning methods. The following DATA step generates 1,000,000 observations:

```
data ex12;
    length id 8;
    do id=1 to 1000000;
        x1 = ranuni(101);
        x2 = 10*ranuni(201);
        output;
    end;
run;
```

You can run PROC HPBIN in distributed mode by specifying valid values for the NODES=, INSTALL=, and HOST= options in the PERFORMANCE statement. An alternative to specifying the INSTALL= and HOST= options in the PERFORMANCE statement is to set appropriate values for the GRIDHOST and

GRIDINSTALLLOC environment variables by using OPTIONS SET commands. See the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics," for details about setting these options or environment variables.

The following statements provide an example. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to macro variables with appropriate values.

```
ods output BinInfo=bininfo;
ods output PerformanceInfo=perfInfo;
ods output Mapping=mapTable;
ods listing close;
proc hpbin data=ex12 output=out numbin=10 quantile;
    input x1-x2;
    performance nodes=4 nthreads=8
    host="&GRIDHOST" install="&GRIDINSTALLLOC";
run;
ods listing;
```

The "Performance Information" table in Output 4.3.1 shows the grid setting.

Output 4.3.1 PROC HPBIN Performance Information

Performance Information	
Host Node	<< your grid host >>
Install Location	<< your grid install location >>
Execution Mode	Distributed
Grid Mode	Symmetric
Number of Compute Nodes	4
Number of Threads per Node	8

The "Binning Information" table in Output 4.3.2 shows the binning method, number of bins, and number of variables.

Output 4.3.2 PROC HPBIN Binning Information

Binning Information	
Method	Quantile Binning
Number of Bins Specified	10
Number of Variables	2

The "Mapping" table in Output 4.3.3 shows the level mapping of the input variables. As you can see from this table, when the binning method is quantile, PROC HPBIN assigns the same number of observations to each bin for the input variables if possible.

Output 4.3.3 PROC HPBIN Mapping

Mapping				
Variable	Binned Variable	Range	Frequency	Proportion
x1	BIN_x1	x1 < 0.099859	100000	0.10000000
		0.099859 <= x1 < 0.199413	100000	0.10000000
		0.199413 <= x1 < 0.299210	100000	0.10000000
		0.299210 <= x1 < 0.399472	100000	0.10000000
		0.399472 <= x1 < 0.499913	100000	0.10000000
		0.499913 <= x1 < 0.599746	100000	0.10000000
		0.599746 <= x1 < 0.700361	100000	0.10000000
		0.700361 <= x1 < 0.800231	100000	0.10000000
		0.800231 <= x1 < 0.900236	100000	0.10000000
		0.900236 <= x1	100000	0.10000000
x2	BIN_x2	x2 < 0.996924	100000	0.10000000
		0.996924 <= x2 < 1.994716	100000	0.10000000
		1.994716 <= x2 < 2.993747	100000	0.10000000
		2.993747 <= x2 < 3.994634	100000	0.10000000
		3.994634 <= x2 < 4.998520	100000	0.10000000
		4.998520 <= x2 < 5.997022	100000	0.10000000
		5.997022 <= x2 < 6.992673	100000	0.10000000
		6.992673 <= x2 < 7.998557	100000	0.10000000
		7.998557 <= x2 < 8.999391	100000	0.10000000
		8.999391 <= x2	100000	0.10000000

Example 4.4: Winsorized Binning

For Winsorized binning, PROC HPBIN provides bucket binning and basic Winsorized statistical information for the input data. The following statements include the WINSOR and WINSORRATE=0.05 options and generate tables for Winsor and trimmed statistics.

```
data ex12;
    length id 8;
    do id=1 to 10000;
        x1 = ranuni(101);
        x2 = 10*ranuni(201);
        x3 = 100*ranuni(301);
        output;
    end;
run;

ods output Winsor=Winsor;
ods output Trim=Trim;
ods listing close;
proc hpbin data=ex12 NUMBIN=10 WINSOR WINSORRATE=0.05;
    input x1-x2;
run;
ods listing;
```

The preceding statements generate Output 4.4.1 and Output 4.4.2.

Output 4.4.1 PROC HPBIN Winsorized Statistics

The HPBIN Procedure

Winsorized Statistics

Variable	Mean	Std Error Mean	N Left Tail	Percent Left Tail	N Right Tail	Percent Right Tail	DF
x1	0.50175743	0.00316436	500	5	501	5.01000000	8998
x2	5.03025502	0.03185024	500	5	501	5.01000000	8998

Output 4.4.2 PROC HPBIN Trimmed Statistics

Trimmed Statistics

Variable	Mean	Std Error Mean	N Left Tail	Percent Left Tail	N Right Tail	Percent Right Tail	DF
x1	0.50209276	0.00316434	500	5	501	5.01000000	8998
x2	5.03201037	0.03185006	500	5	501	5.01000000	8998

Example 4.5: Bucket Binning and Weight-of-Evidence Computation

This example shows bucket binning and weight-of-evidence (WOE) computation in two steps. In the first step, PROC HPBIN does bucket binning and generates the mapping table. In the second step, PROC HPBIN takes the mapping table that is generated in the first step as the BINS_META data set and computes the WOE and the information value (IV).

```
data ex12;
input cl1 $ x0  x1  x2  y $ freq id;
cards;
       a     2    .   7  n   2   1
       a     2    2   6  .   3   2
       a     3    0   1  o   0   3
       c     2    3   7  y   .   4
       c     2    .   4  n   -5  5
       a     3    6   7  n   3   6
       b     1    4   4  y   4   7
       b     2    5   6  y   3   8
       b     1    6   4  o   1   9
       b     2    3   2  n   3   10
run;


 proc hpbin data=ex12 numbin=5;
     input x1/numbin=4;
     input x2;
     ods output Mapping=Mapping;
 run;

 proc hpbin data=ex12 WOE BINS_META=Mapping;
     target y/level=nominal order=desc;
 run;
```

Output 4.5.1 Number of Observations Table

The HPBIN Procedure

Number of Observations Read	10
Number of Observations Used	7

Output 4.5.2 Weight-of-Evidence Table

			Weight of Evidence					
Variable	**Binned Variable**	**Range**	**Non-event Count**	**Non-event Rate**	**Event Count**	**Event Rate**	**Weight of Evidence**	**Information Value**
x1	**BIN_x1**		2	1	0	0	0.91629073	0.30543024
		x1 < 1.500000	1	1	0	0	0.40546511	0.06757752
		1.500000 <= x1 < 3.000000	0	0	0	0	0	0
		3.000000 <= x1 < 4.500000	1	0.33333333	2	0.66666667	-1.3862944	0.69314718
		4.500000 <= x1	2	0.66666667	1	0.33333333	0	0
x2	**BIN_x2**		0	0	0	0	0	0
		x2 < 2.200000	2	1	0	0	0.91629073	0.30543024
		2.200000 <= x2 < 3.400000	0	0	0	0	0	0
		3.400000 <= x2 < 4.600000	2	0.66666667	1	0.33333333	0	0
		4.600000 <= x2 < 5.800000	0	0	0	0	0	0
		5.800000 <= x2	2	0.50000000	2	0.50000000	-0.6931472	0.23104906

Output 4.5.3 Variable Information Value Table

Variable Information Value	
Variable	**Information Value**
x1	1.06615494
x2	0.53647930

Chapter 5
The HPCORR Procedure

Contents

Overview: HPCORR Procedure

The HPCORR procedure computes Pearson correlation coefficients and the probabilities associated with these statistics. The Pearson product-moment correlation is a parametric measure of a linear relationship between two variables.

A related type of correlation, polychoric correlation, measures the correlation between two unobserved variables that have a bivariate normal distribution. Information about these variables is obtained through two corresponding observed ordinal variables that are derived from the unobserved variables by classifying their values into finite sets of discrete, ordered values. Polychoric correlation is not available in the HPCORR procedure, but it is available in the FREQ procedure.

When only one set of analysis variables is specified, the default correlation analysis includes descriptive statistics for each analysis variable and pairwise Pearson correlation statistics for these variables.

When two sets of analysis variables are specified, the default correlation analysis includes descriptive statistics for each analysis variable and pairwise Pearson correlation statistics between the two sets of variables.

When the relationship between two variables is nonlinear or when outliers are present, the correlation coefficient might incorrectly estimate the strength of the relationship.

You can save the correlation statistics in a SAS data set for use with other statistical and reporting procedures.

PROC HPCORR runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Server Distributed Mode.

PROC HPCORR Features

The HPCORR procedure is a high-performance procedure that offers the following functionality:

- It can perform analysis on a massively parallel SAS high-performance appliance.
- It reads input data in parallel and writes output data in parallel when the data source is the appliance database.
- It is highly multithreaded during all phases of analytic execution.

Because the HPCORR procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Getting Started: HPCORR Procedure

This example creates a simple data set and then uses PROC HPCORR to produce simple Pearson correlations by executing on the client machine.

The following statements create the data set Fitness, which has been altered to contain some missing values:

```
*----------------- Data on Physical Fitness -----------------*
| These measurements were made on men involved in a physical |
| fitness course at N.C. State University.                   |
| The variables are Age (years), Weight (kg),                |
| Runtime (time to run 1.5 miles in minutes), and            |
| Oxygen (oxygen intake, ml per kg body weight per minute)   |
| Certain values were changed to missing for the analysis.   |
*------------------------------------------------------------*;
```

```
data Fitness;
   input Age Weight Oxygen RunTime @@;
   datalines;
44 89.47 44.609 11.37    40 75.07 45.313 10.07
44 85.84 54.297  8.65    42 68.15 59.571  8.17
38 89.02 49.874   .      47 77.45 44.811 11.63
40 75.98 45.681 11.95    43 81.19 49.091 10.85
44 81.42 39.442 13.08    38 81.87 60.055  8.63
44 73.03 50.541 10.13    45 87.66 37.388 14.03
45 66.45 44.754 11.12    47 79.15 47.273 10.60
54 83.12 51.855 10.33    49 81.42 49.156  8.95
51 69.63 40.836 10.95    51 77.91 46.672 10.00
48 91.63 46.774 10.25    49 73.37   .    10.08
57 73.37 39.407 12.63    54 79.38 46.080 11.17
52 76.32 45.441  9.63    50 70.87 54.625  8.92
51 67.25 45.118 11.08    54 91.63 39.203 12.88
51 73.71 45.790 10.47    57 59.08 50.545  9.93
49 76.32   .      .      48 61.24 47.920 11.50
52 82.78 47.467 10.50
;
```

The following statements invoke the HPCORR procedure and request a correlation analysis:

```
proc hpcorr data=Fitness;
run;
```

The "Performance Information" table in Figure 5.1 shows that the procedure executes in single-machine mode—that is, the data reside and the computation executes on the machine where the SAS session executes. This run of the HPCORR procedure was performed on a multicore machine; one computational thread was spawned for each core.

The "Simple Statistics" table in Figure 5.1 displays univariate statistics for the analysis variables.

Figure 5.1 Performance Information and Univariate Statistics

The HPCORR Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

4	**Variables:**	Age	Weight	Oxygen	RunTime

Simple Statistics

Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
Age	31	47.67742	5.21144	1478	38.00000	57.00000
Weight	31	77.44452	8.32857	2401	59.08000	91.63000
Oxygen	29	47.22721	5.47718	1370	37.38800	60.05500
RunTime	29	10.67414	1.39194	309.55000	8.17000	14.03000

By default, all numeric variables not listed in other statements are used in the analysis. Observations that have nonmissing values for each variable are used to derive the univariate statistics for that variable.

The "Pearson Correlation Coefficients" table in Figure 5.2 displays the Pearson correlation, the p-value under the null hypothesis of zero correlation, and the number of nonmissing observations for each pair of variables.

Figure 5.2 Pearson Correlation Coefficients

Pearson Correlation Coefficients Prob > |r| under H0: Rho=0 Number of Observations				
	Age	**Weight**	**Oxygen**	**RunTime**
Age	1.00000	-0.23354	-0.31474	0.14478
		0.2061	0.0963	0.4536
	31	31	29	29
Weight	-0.23354	1.00000	-0.15358	0.20072
	0.2061		0.4264	0.2965
	31	31	29	29
Oxygen	-0.31474	-0.15358	1.00000	-0.86843
	0.0963	0.4264		<.0001
	29	29	29	28
RunTime	0.14478	0.20072	-0.86843	1.00000
	0.4536	0.2965	<.0001	
	29	29	28	29

By default, Pearson correlation statistics are computed from observations that have nonmissing values for each pair of analysis variables. Figure 5.2 displays a correlation of −0.86843 between Runtime and Oxygen, which is significant with a *p*-value less than 0.0001. That is, an inverse linear relationship exists between these two variables. As Runtime (time in minutes to run 1.5 miles) increases, Oxygen (oxygen intake in milliliters per kilogram body weight per minute) decreases.

Syntax: HPCORR Procedure

The following statements are available in PROC HPCORR:

PROC HPCORR < *options* > **;**
FREQ *variable* **;**
PERFORMANCE *performance-options* **;**
VAR *variables* **;**
WEIGHT *variable* **;**
WITH *variables* **;**

The FREQ statement specifies the variable that represents the frequency of occurrence for other values in the observation.

The VAR statement lists the numeric variables to be analyzed and their order in the correlation matrix. If you omit the VAR statement, all numeric variables not listed in other statements are used.

The WEIGHT statement identifies the variable whose values weight each observation to compute Pearson product-moment correlation.

The WITH statement lists the numeric variables with which correlations are to be computed.

The PROC HPCORR statement is the only required statement for the HPCORR procedure.

The rest of this section provides detailed syntax information for each of these statements, beginning with the PROC HPCORR statement. The remaining statements are presented in alphabetical order.

PROC HPCORR Statement

PROC HPCORR < *options* > **;**

Table 5.1 summarizes the *options* available in the PROC HPCORR statement.

Table 5.1 Summary of PROC HPCORR Options

Option	Description
Data Sets	
DATA=	Specifies the input data set
OUTP=	Specifies the output data set with Pearson correlation statistics
Statistical Analysis	
EXCLNPWGT	Excludes from the analysis observations that have nonpositive weight values
NOMISS	Excludes from the analysis observations that have missing analysis values
Pearson Correlation Statistics	
COV	Computes covariances
CSSCP	Computes corrected sums of squares and crossproducts
SSCP	Computes sums of squares and crossproducts
Printed Output	
BEST=	Displays the specified number of ordered correlation coefficients
NOCORR	Suppresses Pearson correlations
NOPRINT	Suppresses all printed output
NOPROB	Suppresses *p*-values
NOSIMPLE	Suppresses descriptive statistics
RANK	Displays ordered correlation coefficients

You can specify the following *options* in the PROC HPCORR statement:

BEST=*n*

prints the *n* highest correlation coefficients for each variable. The value of *n* must be greater than or equal to 1. Correlations are ordered from highest to lowest in absolute value. If you do not specify this option, PROC HPCORR prints correlations in a rectangular table, using the variable names as row and column labels.

COV

displays the variance/covariance matrix. When you specify this option, the Pearson correlations are also displayed. If you specify the OUTP= option, the output data set also contains the covariance matrix with the corresponding _TYPE_ variable value `'COV'`.

CSSCP

displays a table of the corrected sums of squares and crossproducts. When you specify this option, the Pearson correlations are also displayed. If you specify the OUTP= option, the output data set also contains a CSSCP matrix with the corresponding _TYPE_ variable value `'CSSCP'`.

DATA=*SAS-data-set*
names the SAS data set to be analyzed by PROC HPCORR. If you do not specify this option, PROC HPCORR uses the most recently created SAS data set.

EXCLNPWGT
EXCLNPWGTS
excludes from the analysis observations that have nonpositive weight values. If you do not specify this option, PROC HPCORR treats observations with negative weights like those with zero weights and counts them in the total number of observations.

NOCORR
suppresses the display of the Pearson correlations. If you specify the OUTP= option, the data set _TYPE_ remains `'CORR'`. To change the data set type to COV, CSSCP, or SSCP, use the (TYPE=) data set option.

NOMISS
excludes from the analysis observations that have missing values. If you do not specify this option, PROC HPCORR computes correlation statistics by using all of the nonmissing pairs of variables. Using this option is computationally more efficient.

NOPRINT
suppresses all displayed output. Use this option if you only want to create an output data set.

NOPROB
suppresses the display of the probabilities that are associated with each correlation coefficient.

NOSIMPLE
suppresses the printing of simple descriptive statistics for each variable. However, if you request an output data set, the output data set still contains simple descriptive statistics for the variables.

OUTP=*output-data-set*
OUT=*output-data-set*
creates an output data set that contains Pearson correlation statistics. This data set also includes means, standard deviations, and the number of observations. The value of the _TYPE_ variable is `'CORR'`.

RANK
displays the ordered correlation coefficients for each variable. Correlations are ordered from highest to lowest in absolute value.

SSCP
displays a table of the sums of squares and crossproducts. When you specify this option, the Pearson correlations are also displayed. If you specify the OUTP= option, the output data set contains a SSCP matrix and the corresponding _TYPE_ variable value is `'SSCP'`.

VARDEF=DF | N | WDF | WEIGHT | WGT
specifies the variance divisor in the calculation of variances and covariances. The default is VARDEF=DF.

Table 5.2 shows the available values and associated divisors for the VARDEF= option, where n is the number of nonmissing observations and w_j is the weight associated with the jth nonmissing observation.

Table 5.2 Possible Values for the VARDEF= Option

Value	Description	Divisor
DF	Degrees of freedom	$n - 1$
N	Number of observations	n
WDF	Sum of weights minus one	$\sum_j^n w_j - 1$
WEIGHT | WGT	Sum of weights	$\sum_j^n w_j$

FREQ Statement

FREQ *variable* **;**

The FREQ statement specifies a numeric *variable* whose value represents the frequency of the observation. If you use the FREQ statement, PROC HPCORR assumes that each observation represents n observations, where n is the value of the FREQ *variable*. If n is not an integer, SAS truncates it. If n is less than 1 or is missing, the observation is excluded from the analysis. The sum of the frequency variables represents the total number of observations.

The effects of the FREQ and WEIGHT statements are similar except when the degrees of freedom are calculated.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of the HPCORR procedure.

You can also use the PERFORMANCE statement to control whether the HPCORR procedure executes in single-machine mode or distributed mode.

The PERFORMANCE statement is documented further in the section "PERFORMANCE Statement" on page 36.

VAR Statement

VAR *variables* **;**

The VAR statement lists variables for which correlation coefficients are to be computed. If the VAR statement is not specified, PROC HPCORR computes correlations for all numeric variables that are not listed in other statements.

WEIGHT Statement

WEIGHT *variable* **;**

The WEIGHT statement lists weights to use in the calculation of Pearson weighted product-moment correlation.

Observations that have missing weights are excluded from the analysis. By default, for observations that have nonpositive weights, weights are set to 0 and the observations are included in the analysis. You can use the EXCLNPWGT option to exclude observations with negative or zero weights from the analysis.

WITH Statement

WITH *variables* **;**

The WITH statement lists variables with which correlations of the VAR statement variables are to be computed. The WITH statement requests correlations of the form $r(X_i, Y_j)$, where $X_1, \ldots, X_m$ are analysis variables that you specify in the VAR statement and $Y_1, \ldots, Y_n$ are variables that you specify in the WITH statement. The correlation matrix has a rectangular structure of the form

$$\begin{bmatrix} r(Y_1, X_1) & \cdots & r(Y_1, X_m) \\ \vdots & \ddots & \vdots \\ r(Y_n, X_1) & \cdots & r(Y_n, X_m) \end{bmatrix}$$

For example, the statements

```
proc corr;
   var x1 x2;
   with y1 y2 y3;
run;
```

produce correlations for the following combinations:

$$\begin{bmatrix} r(Y1, X1) & r(Y1, X2) \\ r(Y2, X1) & r(Y2, X2) \\ r(Y3, X1) & r(Y3, X2) \end{bmatrix}$$

Details: HPCORR Procedure

Pearson Product-Moment Correlation

The Pearson product-moment correlation is a parametric measure of association for two variables. It measures both the strength and the direction of a linear relationship. If one variable X is an exact linear function of another variable Y, a positive relationship exists if the correlation is 1, and a negative relationship exists if the correlation is -1. If there is no linear predictability between the two variables, the correlation is 0. If the

two variables are normal with a correlation 0, the two variables are independent. Correlation does not imply causality because, in some cases, an underlying causal relationship might not exist.

The formula for the population Pearson product-moment correlation, denoted ρ_{xy}, is

$$\rho_{xy} = \frac{\mathrm{Cov}(x,y)}{\sqrt{\mathrm{V}(x)\mathrm{V}(y)}} = \frac{\mathrm{E}(\,(x-\mathrm{E}(x))(y-\mathrm{E}(y))\,)}{\sqrt{\mathrm{E}(x-\mathrm{E}(x))^2\,\mathrm{E}(y-\mathrm{E}(y))^2}}$$

The sample correlation, such as a Pearson product-moment correlation or weighted product-moment correlation, estimates the population correlation. The formula for the sample Pearson product-moment correlation is as follows, where $\bar{x}$ is the sample mean of x is the sample mean of y:

$$r_{xy} = \frac{\sum_i (\,(x_i-\bar{x})(y_i-\bar{y})\,)}{\sqrt{\sum_i (x_i-\bar{x})^2 \sum_i (y_i-\bar{y})^2}}$$

The formula for a weighted Pearson product-moment correlation is as follows, where w_i is the weight, $\bar{x}_w$ is the weighted mean of x, and $\bar{y}_w$ is the weighted mean of y:

$$r_{xy} = \frac{\sum_i w_i (x_i-\bar{x}_w)(y_i-\bar{y}_w)}{\sqrt{\sum_i w_i (x_i-\bar{x}_w)^2 \sum_i w_i (y_i-\bar{y}_w)^2}}$$

Probability Values

Probability values for the Pearson correlation are computed by treating the following equation as if it came from a t distribution with $(n-2)$ degrees of freedom, where r is the sample correlation:

$$t = (n-2)^{1/2}\left(\frac{r^2}{1-r^2}\right)^{1/2}$$

The partial variance-covariance matrix is calculated with the variance divisor (specified in the VARDEF= option). PROC HPCORR then uses the standard Pearson correlation formula on the partial variance-covariance matrix to calculate the Pearson partial correlation matrix.

When a correlation matrix is positive definite, the resulting partial correlation between variables x and y after adjusting for a single variable z is identical to that obtained from the following first-order partial correlation formula, where r_{xy}, r_{xz}, and r_{yz} are the appropriate correlations:

$$r_{xy.z} = \frac{r_{xy} - r_{xz}r_{yz}}{\sqrt{(1-r_{xz}^2)(1-r_{yz}^2)}}$$

The formula for higher-order partial correlations is a straightforward extension of the preceding first-order formula. For example, when the correlation matrix is positive definite, the partial correlation between x and y that controls for both z_1 and z_2 is identical to the following second-order partial correlation formula, where $r_{xy.z_1}$, $r_{xz_2.z_1}$, and $r_{yz_2.z_1}$ are first-order partial correlations among variables x, y, and z_2 given z_1:

$$r_{xy.z_1z_2} = \frac{r_{xy.z_1} - r_{xz_2.z_1}r_{yz_2.z_1}}{\sqrt{(1-r_{xz_2.z_1}^2)(1-r_{yz_2.z_1}^2)}}$$

Multithreading

Threading refers to the organization of computational work into multiple tasks (processing units that can be scheduled by the operating system). A task is associated with a thread. Multithreading refers to the concurrent execution of threads. When multithreading is possible, substantial performance gains can be realized compared to sequential (single-threaded) execution.

The number of threads spawned by the HPCORR procedure is determined by the number of CPUs on a machine and can be controlled in the following ways:

- You can specify the CPU count with the CPUCOUNT= SAS system option. For example, if you specify the following statements, the HPCORR procedure schedules threads as if it executed on a system with four CPUs, regardless of the actual CPU count:

```
options cpucount=4;
```

- You can specify the NTHREADS= option in the PERFORMANCE statement to determine the number of threads. This specification overrides the NOTHREADS system option. Specify NTHREADS=1 to force single-threaded execution.

The number of threads per machine is displayed in the "Performance Information" table, which is part of the default output. The HPCORR procedure allocates one thread per CPU.

The HPCORR procedure implements a data-parallel model. For example, if the input data set has 1,000 observations and you are running with four threads, then 250 observations are associated with each thread. All operations that require access to the data are then multithreaded.

Output Tables

By default, PROC HPCORR prints a report that includes descriptive statistics and correlation statistics for each variable. The descriptive statistics include the number of observations that have nonmissing values, the mean, the standard deviation, the minimum, and the maximum.

If a nonparametric measure of association is requested, the descriptive statistics include the median. Otherwise, the sample sum is included. If a Pearson partial correlation is requested, the descriptive statistics also include the partial variance and partial standard deviation.

If variable labels are available, PROC HPCORR labels the variables. If you specify the CSSCP, SSCP, or COV option, the appropriate sums of squares and crossproducts and covariance matrix appear at the top of the correlation report. If the data set contains missing values, PROC HPCORR prints additional statistics for each pair of variables.

These statistics, calculated from the observations that have nonmissing row and column variable values, might include the following:

- SSCP('W','V'), uncorrected sums of squares and crossproducts

- USS(‘W’), uncorrected sums of squares for the row variable
- USS(‘V’), uncorrected sums of squares for the column variable
- CSSCP(‘W’,‘V’), corrected sums of squares and crossproducts
- CSS(‘W’), corrected sums of squares for the row variable
- CSS(‘V’), corrected sums of squares for the column variable
- COV(‘W’,‘V’), covariance
- VAR(‘W’), variance for the row variable
- VAR(‘V’), variance for the column variable
- DF(‘W’,‘V’), divisor for calculating covariance and variances

For each pair of variables, PROC HPCORR prints the correlation coefficients, the number of observations that are used to calculate the coefficient, and the *p*-value.

Output Data Sets

PROC HPCORR creates an output data set that contains statistics for Pearson correlation. By default, the output data set is a special data set type (TYPE=CORR) that is recognized by many SAS/STAT procedures, including the REG and FACTOR procedures. When you specify the NOCORR option and the COV, CSSCP, or SSCP option, use the (TYPE=) data set option to change the data set type to COV, CSSCP, or SSCP.

The output data set includes the following variables:

- BY variables, which identify the BY group when a BY statement is used
- _TYPE_ variable, which identifies the type of observation
- _NAME_ variable, which identifies the variable that corresponds to a given row of the correlation matrix
- INTERCEPT variable, which identifies variable sums when the SSCP option is specified
- VAR variables, which identify the variables listed in the VAR statement

You can use a combination of the _TYPE_ and _NAME_ variables to identify the contents of an observation. The _NAME_ variable indicates which row of the correlation matrix the observation corresponds to. The values of the _TYPE_ variable are as follows:

- SSCP, uncorrected sums of squares and crossproducts
- CSSCP, corrected sums of squares and crossproducts
- COV, covariances

- MEAN, mean of each variable
- STD, standard deviation of each variable
- N, number of nonmissing observations for each variable
- SUMWGT, sum of the weights for each variable when using a WEIGHT statement
- HPCORR, correlation statistics for each variable

If you specify the SSCP option, the OUTP= data set includes an additional observation that contains intercept values.

ODS Table Names

PROC HPCORR assigns a name to each table it creates. These names are listed in Table 5.3. You must use these names to refer to tables when you use the Output Delivery System (ODS). For more information about ODS, see Chapter 20, "Using the Output Delivery System" (*SAS/STAT User's Guide*).

Table 5.3 ODS Tables Produced by PROC HPCORR

ODS Table Name	Description	Option
Cov	Covariances	COV
Csscp	Corrected sums of squares and crossproducts	CSSCP
PearsonCorr	Pearson correlations	
SimpleStats	Simple descriptive statistics	
Sscp	Sums of squares and crossproducts	SSCP
VarInformation	Variable information	
PerformanceInfo	Information about high-performance computing environment	

Examples: HPCORR Procedure

Example 5.1: Computing the Pearson Measure of Association in Single-Machine Mode

The Fitness data set created in the section "Getting Started: HPCORR Procedure" on page 68 contains measurements from a study of physical fitness of 31 participants. The following statements request the Pearson measure of association for the variables Weight, Oxygen, and Runtime:

```
title 'Measures of Association for a Physical Fitness Study';
proc hpcorr data=Fitness pearson;
   var Weight Oxygen RunTime;
run;
```

The "Simple Statistics" table in Output 5.1.1 displays univariate descriptive statistics for the analysis variables. By default, observations that have nonmissing values for each variable are used to derive the univariate statistics for that variable.

Output 5.1.1 Simple Statistics

Measures of Association for a Physical Fitness Study

The HPCORR Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

3	**Variables:**	Weight	Oxygen	RunTime

Simple Statistics

Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
Weight	31	77.44452	8.32857	2401	59.08000	91.63000
Oxygen	29	47.22721	5.47718	1370	37.38800	60.05500
RunTime	29	10.67414	1.39194	309.55000	8.17000	14.03000

The "Pearson Correlation Coefficients" table in Output 5.1.2 displays the Pearson correlation statistics for pairs of analysis variables. The Pearson correlation is a parametric measure of association for two continuous random variables. When the data have missing values, the number of observations used to calculate the correlation can vary.

Output 5.1.2 Pearson Correlation Coefficients

Pearson Correlation Coefficients
Prob > |r| under H0: Rho=0
Number of Observations

	Weight	Oxygen	RunTime
Weight	1.00000	-0.15358	0.20072
		0.4264	0.2965
	31	29	29
Oxygen	-0.15358	1.00000	-0.86843
	0.4264		<.0001
	29	29	28
RunTime	0.20072	-0.86843	1.00000
	0.2965	<.0001	
	29	28	29

The table shows that the Pearson correlation between Runtime and Oxygen is -0.86843, which is significant with a p-value less than 0.0001. This indicates a strong negative linear relationship between these two variables. As Runtime increases, Oxygen decreases linearly.

Example 5.2: Computing the Pearson Measure of Association in Distributed Mode

The real power of PROC HPCORR is when the computation is solved with multiple threads or in a distributed environment.

You can switch to running in distributed mode simply by specifying valid values for the NODES=, INSTALL=, and HOST= options in the PERFORMANCE statement.

An alternative to specifying the INSTALL= and HOST= options in the PERFORMANCE statement is to set appropriate values for the GRIDHOST and GRIDINSTALLLOC environment variables by using OPTIONS SET commands. For more information about setting these options or environment variables, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

The following statements provide an example. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to macro variables with appropriate values.

The macro variable BRECLIB is the name of a libref to a billion-record database.

```
title 'PROC HPCORR Processing Billion-Record Database';
proc hpcorr data=&BRECLIB;
   var x1-x5;
   performance host=&GRIDHOST install=&GRIDINSTALLLOC;
run;
```

The execution mode in the "Performance Information" table shown in Output 5.2.1 indicates that the calculations were performed in a distributed environment that uses 16 nodes; the data are predistributed using a Greenplum parallel database.

Output 5.2.1 Performance Information in Distributed Mode

PROC HPCORR Processing Billion-Record Database

Performance Information	
Host Node	<< your grid host >>
Install Location	<< your grid install location >>
Execution Mode	Distributed
Number of Compute Nodes	16

Another indication of distributed execution is the following message, which is issued by all high-performance analytical procedures in the SAS log:

```
NOTE: The HPCORR procedure is executing in the distributed
      computing environment with 16 worker nodes.
```

Because the sample database uses random data, the results are not meaningful. The power of high-performance analytics is that this test can be completed in a matter of minutes instead of hours.

References

Anderson, T. W. (1984), *An Introduction to Multivariate Statistical Analysis*, 2nd Edition, New York: John Wiley & Sons.

Blum, J. R., Kiefer, J., and Rosenblatt, M. (1961), "Distribution Free Tests of Independence Based on the Sample Distribution Function," *Annals of Mathematical Statistics*, 32, 485–498.

Conover, W. J. (1999), *Practical Nonparametric Statistics*, 3rd Edition, New York: John Wiley & Sons.

Cox, N. R. (1974), "Estimation of the Correlation between a Continuous and a Discrete Variable," *Biometrics*, 30, 171–178.

Cronbach, L. J. (1951), "Coefficient Alpha and the Internal Structure of Tests," *Psychometrika*, 16, 297–334.

Drasgow, F. (1986), "Polychoric and Polyserial Correlations," in S. Kotz, N. L. Johnson, and C. B. Read, eds., *Encyclopedia of Statistical Sciences*, volume 7, 68–74, New York: John Wiley & Sons.

Fisher, R. A. (1915), "Frequency Distribution of the Values of the Correlation Coefficient in Samples from an Indefinitely Large Population," *Biometrika*, 10, 507–521.

Fisher, R. A. (1921), "On the 'Probable Error' of a Coefficient of Correlation Deduced from a Small Sample," *Metron*, 1, 3–32.

Fisher, R. A. (1936), "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, 7, 179–188.

Fisher, R. A. (1970), *Statistical Methods for Research Workers*, 14th Edition, Davien, CT: Hafner Publishing.

Hoeffding, W. (1948), "A Non-parametric Test of Independence," *Annals of Mathematical Statistics*, 19, 546–557.

Hollander, M. and Wolfe, D. A. (1999), *Nonparametric Statistical Methods*, 2nd Edition, New York: John Wiley & Sons.

Keeping, E. S. (1962), *Introduction to Statistical Inference*, New York: D. Van Nostrand.

Knight, W. E. (1966), "A Computer Method for Calculating Kendall's Tau with Ungrouped Data," *Journal of the American Statistical Association*, 61, 436–439.

Moore, D. S. (2000), *Statistics: Concepts and Controversies*, 5th Edition, New York: W. H. Freeman.

Mudholkar, G. S. (1983), "Fisher's z-Transformation," *Encyclopedia of Statistical Sciences*, 3, 130–135.

Noether, G. E. (1967), *Elements of Nonparametric Statistics*, New York: John Wiley & Sons.

Novick, M. R. (1967), "Coefficient Alpha and the Reliability of Composite Measurements," *Psychometrika*, 32, 1–13.

Nunnally, J. C. and Bernstein, I. H. (1994), *Psychometric Theory*, 3rd Edition, New York: McGraw-Hill.

Olsson, U., Drasgow, F., and Dorans, N. J. (1982), "The Polyserial Correlation Coefficient," *Biometrika*, 47, 337–347.

Ott, R. L. and Longnecker, M. T. (2000), *An Introduction to Statistical Methods and Data Analysis*, 5th Edition, Belmont, CA: Wadsworth.

Spector, P. E. (1992), *Summated Rating Scale Construction: An Introduction*, Newbury Park, CA: Sage Publications.

Yu, C. H. (2001), "An Introduction to Computing and Interpreting Cronbach Coefficient Alpha in SAS," in *Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference*, Cary, NC: SAS Institute Inc.

Chapter 6
The HPDMDB Procedure

Contents

Overview: HPDMDB Procedure

The HPDMDB procedure is a high-performance version of the DMDB procedure, which creates summaries of the input data source. PROC HPDMDB creates two output data sets: the VAROUT data set, which contains a summary of the numeric variables, and the CLASSOUT data set, which contains a summary of the classification variables.

PROC HPDMDB is high-performance in that it takes advantage of distributed and multicore computing environments when the input data are stored on the SAS appliance.

You can use PROC HPDMDB to create a data mining database (DMDB) that is compatible with the DMDB from PROC DMDB, although this feature of PROC HPDMDB might not be supported in future versions.

PROC HPDMDB runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Server Distributed Mode.

Because the HPDMDB procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations

- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Getting Started: HPDMDB Procedure

The HPDMDB procedure summarizes data. The following example uses the Sampsio.Hmeq data set, which includes information about 5,960 fictitious mortgages. Each case represents an applicant for a home equity loan, and all applicants have an existing mortgage. The binary target variable BAD indicates whether an applicant eventually defaulted or was ever seriously delinquent. There are 10 numeric input variables and three classification input variables.

```
proc hpdmdb data=Sampsio.Hmeq
    classout=cout varout=vout;

    var loan derog mortdue value yoj delinq
        clage ninq clno debtinc;
    class bad(desc) reason(ascending) job;
run;

proc print data=cout;run;
proc print data=vout;run;
```

The data set cout (shown in Figure 6.1) contains the class summary table with levels sorted according to the sort option in the CLASS statement of PROC HPDMDB. You can see that the levels for BAD are in descending order and the levels for REASON are in ascending order. The levels for JOB are in the default ascending order.

Figure 6.1 Summaries of Classification Variables in Sampsio.Hmeq Data Set

The HPDMDB Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Data Access Information			
Data	**Engine**	**Role**	**Path**
WORK.COUT	V9	Output	On Client
WORK.VOUT	V9	Output	On Client
SAMPSIO.HMEQ	V9	Input	On Client

Figure 6.1 *continued*

Obs	NAME	LEVEL	CODE	FREQUENCY	TYPE	CRAW	NRAW	FREQPERCENT	NMISSPERCENT
1	BAD	1	1	1189	N		1	19.9497	19.9497
2	BAD	0	0	4771	N		0	80.0503	80.0503
3	REASON		0	252	C		.	4.2282	.
4	REASON	DEBTCON	1	3928	C	DebtCon	.	65.9060	68.8157
5	REASON	HOMEIMP	2	1780	C	HomeImp	.	29.8658	31.1843
6	JOB		0	279	C		.	4.6812	.
7	JOB	MGR	1	767	C	Mgr	.	12.8691	13.5011
8	JOB	OFFICE	2	948	C	Office	.	15.9060	16.6872
9	JOB	OTHER	3	2388	C	Other	.	40.0671	42.0349
10	JOB	PROFEXE	4	1276	C	ProfExe	.	21.4094	22.4608
11	JOB	SALES	5	109	C	Sales	.	1.8289	1.9187
12	JOB	SELF	6	193	C	Self	.	3.2383	3.3973

Obs	NAME	NMISS	N	MIN	MAX	MEAN	STD	SKEWNESS	KURTOSIS
1	LOAN	0	5960	1100.00	89900.00	18607.97	11207.48	2.02378	6.9326
2	DEROG	708	5252	0.00	10.00	0.25	0.85	5.32087	36.8728
3	MORTDUE	518	5442	2063.00	399550.00	73760.82	44457.61	1.81448	6.4819
4	VALUE	112	5848	8000.00	855909.00	101776.05	57385.78	3.05334	24.3628
5	YOJ	515	5445	0.00	41.00	8.92	7.57	0.98846	0.3721
6	DELINQ	580	5380	0.00	15.00	0.45	1.13	4.02315	23.5654
7	CLAGE	308	5652	0.00	1168.23	179.77	85.81	1.34341	7.5995
8	NINQ	510	5450	0.00	17.00	1.19	1.73	2.62198	9.7865
9	CLNO	222	5738	0.00	71.00	21.30	10.14	0.77505	1.1577
10	DEBTINC	1267	4693	0.52	203.31	33.78	8.60	2.85235	50.5040

Obs	SUM	USS	CSS
1	110903500.00	2.8121848E12	748495791434.56
2	1337.00	4099.00	3758.64
3	401406367.20	4.0362084E13	10754022449877
4	595186333.04	7.9830628E13	19254914800672
5	48581.75	745755.59	312296.19
6	2418.00	7922.00	6835.25
7	1016038.99	224259958.52	41610414.32
8	6464.00	23950.00	16283.34
9	122197.00	3192071.00	589751.93
10	158529.14	5702262.28	347161.26

Numeric summaries are in the data set **vout**, shown in Figure 6.2.

Figure 6.2 Summaries of Numeric Variables in Sampsio.Hmeq Data Set

The HPDMDB Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Data Access Information			
Data	**Engine**	**Role**	**Path**
WORK.COUT	V9	Output	On Client
WORK.VOUT	V9	Output	On Client
SAMPSIO.HMEQ	V9	Input	On Client

Obs	NAME	LEVEL	CODE	FREQUENCY	TYPE	CRAW	NRAW	FREQPERCENT	NMISSPERCENT
1	BAD	1	1	1189	N		1	19.9497	19.9497
2	BAD	0	0	4771	N		0	80.0503	80.0503
3	REASON		0	252	C		.	4.2282	.
4	REASON	DEBTCON	1	3928	C	DebtCon	.	65.9060	68.8157
5	REASON	HOMEIMP	2	1780	C	HomeImp	.	29.8658	31.1843
6	JOB		0	279	C		.	4.6812	.
7	JOB	MGR	1	767	C	Mgr	.	12.8691	13.5011
8	JOB	OFFICE	2	948	C	Office	.	15.9060	16.6872
9	JOB	OTHER	3	2388	C	Other	.	40.0671	42.0349
10	JOB	PROFEXE	4	1276	C	ProfExe	.	21.4094	22.4608
11	JOB	SALES	5	109	C	Sales	.	1.8289	1.9187
12	JOB	SELF	6	193	C	Self	.	3.2383	3.3973

Figure 6.2 *continued*

Obs	NAME	NMISS	N	MIN	MAX	MEAN	STD	SKEWNESS	KURTOSIS
1	LOAN	0	5960	1100.00	89900.00	18607.97	11207.48	2.02378	6.9326
2	DEROG	708	5252	0.00	10.00	0.25	0.85	5.32087	36.8728
3	MORTDUE	518	5442	2063.00	399550.00	73760.82	44457.61	1.81448	6.4819
4	VALUE	112	5848	8000.00	855909.00	101776.05	57385.78	3.05334	24.3628
5	YOJ	515	5445	0.00	41.00	8.92	7.57	0.98846	0.3721
6	DELINQ	580	5380	0.00	15.00	0.45	1.13	4.02315	23.5654
7	CLAGE	308	5652	0.00	1168.23	179.77	85.81	1.34341	7.5995
8	NINQ	510	5450	0.00	17.00	1.19	1.73	2.62198	9.7865
9	CLNO	222	5738	0.00	71.00	21.30	10.14	0.77505	1.1577
10	DEBTINC	1267	4693	0.52	203.31	33.78	8.60	2.85235	50.5040

Obs	SUM	USS	CSS
1	110903500.00	2.8121848E12	748495791434.56
2	1337.00	4099.00	3758.64
3	401406367.20	4.0362084E13	10754022449877
4	595186333.04	7.9830628E13	19254914800672
5	48581.75	745755.59	312296.19
6	2418.00	7922.00	6835.25
7	1016038.99	224259958.52	41610414.32
8	6464.00	23950.00	16283.34
9	122197.00	3192071.00	589751.93
10	158529.14	5702262.28	347161.26

Syntax: HPDMDB Procedure

The following statements are available in the HPDMDB procedure:

PROC HPDMDB DATA= < *libref.*>*SAS-data-set* < *options*> ;
CLASS *variable* (< *order-option*>) *variable* (< *order-option*>) . . . ;
FREQ *variable* ;
PERFORMANCE *performance-options* ;
VAR *variables* ;
WEIGHT *variable* ;

PROC HPDMDB Statement

PROC HPDMDB DATA= < *libref.*>*SAS-data-set* < *options*> ;

The PROC HPDMDB statement invokes the procedure.

Required Arguments

DATA=< *libref.*>*SAS-data-set*
names the SAS data set that contains the information that you want added to the data mining database. If the data set resides on the SAS appliance, then the SAS appliance is used during summarization.

Optional Arguments

DMDBCAT=< *libref.*>*SAS-catalog*
names the metadata catalog to be created by PROC HPDMDB.

CLASSOUT=< *libref.*>*SAS-data-set*
names the data set to contain the summaries of classification variables that are specified in the CLASS statement.

VAROUT=< *libref.*>*SAS-data-set*
names the data set to contain the summaries of analysis variables that are specified in the VAR statement.

VARDEF=*divisor*
specifies the divisor to use in the calculation of the variance and standard deviation. Table 6.1 shows the possible values for *divisor*.

Table 6.1 Values for *divisor*

Value	Divisor
N	Number of observations (n)
DF (default)	Degrees of freedom ($n - 1$)

PRINT
prints the class-level information to all open ODS destinations.

SPECIALMISSING
enables special missing values to be treated as separate levels.

MAXLEVEL=*max*
specifies the maximum number of levels to be reported for each class variable. If more than *max* levels of a class variable exist, PROC HPDMDB reports the frequency of unreported observations in a level named *_OTHER_*.

CLASS Statement

CLASS *variable (< order-option >) < variable (< order-option >) > . . .* **;**

The CLASS statement specifies the variables whose values define subgroup combinations for the analysis.

The CLASS and VAR statements are mutually exclusive.

Required Argument

variable
specifies a classification variable to be used in the analysis. For each level of a CLASS variable, the CLASSOUT data set contains information about each of the following: the level value, its frequency, and the type of the variable (numeric or character).

Optional Argument

order-option
specifies the order to use when considering the levels of CLASS variables to be sorted. The value of *order-option* can be one of the following:

ASCENDING | ASC
arranges class levels in lowest-to-highest order of unformatted values.

DESCENDING | DESC
arranges class levels in highest-to-lowest order of unformatted values.

ASCFORMATTED | ASCFMT
arranges class levels in ascending order by their formatted values.

DESFORMATTED | DESFMT
arranges class levels in descending order by their formatted values.

DSORDER | DATA
arranges class levels according to the order of their appearance in the input data set.

NOTE: The DSORDER sort option is not supported for input data sets that are stored on the SAS appliance.

FREQ Statement

FREQ *variable* **;**

The FREQ statement specifies a numeric *variable* that contains the frequency of each observation.

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. For observations where *variable* is 0 or missing, the observation is omitted in the CLASSOUT data set and is not included in statistical calculations.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS high-performance analytical procedure.

With the PERFORMANCE statement you can also control whether a SAS high-performance analytical procedure executes in single-machine mode or distributed mode.

The PERFORMANCE statement for SAS high-performance analytical procedures is documented in the section "PERFORMANCE Statement" on page 36 of Chapter 3, "Shared Concepts and Topics."

VAR Statement

VAR *variables* **;**

The VAR statement specifies one or more numeric *variables* to analyze. The CLASS and VAR statements are mutually exclusive.

Required Argument

variables

specifies one or more numeric *variables* to be used in the analysis. The *variables* must be numeric. For each *variable*, the VAROUT data set and the metadata contain the following statistics:

Table 6.2 Statistics Recorded for VAR Variables

Statistic	Meaning
N	Number of observations
NMISS	Number of observations that contain a missing value
MIN	Minimum observed value
MAX	Maximum observed value
MEAN	Mean of observed values
STD	Standard deviation
SKEWNESS	Measure of asymmetry
KURTOSIS	Measure of the "heaviness of the tails"
SUM	Sum of all nonmissing observations
CSS	Corrected sum of squares
USS	Sum of squares

(See Appendix 1, "SAS Elementary Statistics Procedures" in *Base SAS Procedures Guide*, for formulas and other details.)

WEIGHT Statement

WEIGHT *variable* **;**

The WEIGHT statement specifies a numeric *variable* that contains a weight for each observation. The *variable* is used in the computation of means, standard deviation, skewness, and kurtosis.

Required Argument

variable

represents how the observation should be weighted in statistical calculations. For observations where *variable* is 0 or missing, the observation is still included in the CLASSOUT data set but the value is not used in statistical calculations.

Details: HPDMDB Procedure

The statistics recorded for numeric variables are detailed in the section "VAR Statement" on page 90.

For classification variables, a *level* is a distinct observed value after formatting, removal of beginning and ending white space, and capitalization. For example, the values **`MyLevel`** and **`MYLEVEL`** are treated as a single level in the data set. Classification variables can be numeric, and the same levelization rules apply. For example, **`3.000002`** and **`3.0000001`** are treated as the same level if they are formatted using **`BEST3`**.

Frequencies should be integers. If a noninteger frequency is specified, it is rounded to the nearest integer for calculations. Weights do not need to be integers. Negative frequencies and weights are treated as 0.

Examples: HPDMDB Procedure

Example 6.1: Running PROC HPDMDB on the Client

This example demonstrates how to run the HPDMDB procedure on the following data set on the client:

```
data ex;
input x1 x2 x3 x4 y$ w f y2;
cards;
1  2 1 1  m .90 1 0
1  2 1 2  m .91 2 1
1  2 1 3  x .89 1 4
1  2 1 4  x .90 2 4
1  3 1 1  m .91 1 1
1  3 1 2  m .89 2 1
2  3 1 3  x .90 1 5
2  3 1 4  x .89 2 5
3  1 2 1  z .90 1 2
3  1 2 2  z .89 2 2
3  1 2 3  y .90 1 7
3  1 2 4  y .89 2 7
3  4 2 1  z .90 1 3
3  4 2 2  z .89 2 3
4  4 2 3  y .90 1 6
4  4 2 4  y .89 2 6
;
run;
```

When the input data set resides on the client and no PERFORMANCE statement is specified, as in the following example, the client performs all computations:

```
proc hpdmdb data=ex print classout=cout varout=vout;
    class x1-x3;
    weight w;
    var x4 y2;
    freq f;
run;
```

Output 6.1.1 shows the summaries of the numeric variables in the data set ex.

Output 6.1.1 Summaries of Numeric Variables in ex Data Set

Obs	NAME	NMISS	N	MIN	MAX	MEAN	STD	SKEWNESS	KURTOSIS	SUM	USS	CSS
1	x4	0	24	1	4	2.66326	1.06938	-0.05848	-1.19599	57.26	178.80	26.302
2	y2	0	24	0	7	3.57721	2.10658	0.12214	-1.04243	76.91	377.19	102.067

Output 6.1.2 shows the summaries of the classification variables in the data set ex.

Output 6.1.2 Summaries of Classification Variables in ex Data Set

Obs	NAME	LEVEL	CODE	FREQUENCY	TYPE	CRAW	NRAW	FREQPERCENT	NMISSPERCENT
1	x1	1	0	9	N		1	37.5	37.5
2	x1	2	1	3	N		2	12.5	12.5
3	x1	3	2	9	N		3	37.5	37.5
4	x1	4	3	3	N		4	12.5	12.5
5	x2	1	0	6	N		1	25.0	25.0
6	x2	2	1	6	N		2	25.0	25.0
7	x2	3	2	6	N		3	25.0	25.0
8	x2	4	3	6	N		4	25.0	25.0
9	x3	1	0	12	N		1	50.0	50.0
10	x3	2	1	12	N		2	50.0	50.0

Example 6.2: Running with Client Data on the SAS appliance

This example uses the same data set as in Example 6.1.

When the input data set resides on the client and a PERFORMANCE statement with a NODES= option is specified, as in the following example, PROC HPDMDB copies the data set to the SAS appliance, which does the computations. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the macro variable references in the example with the appropriate values.

```
option set=GRIDHOST       = "&GRIDHOST";
option set=GRIDINSTALLLOC = "&GRIDINSTALLLOC";
/*Perform the computation on the SAS appliance using 5 nodes*/
proc hpdmdb data=ex print classout=cout varout=vout;
    class x1-x3;
    weight w;
    var x4 y2;
    freq f;
    performance nodes=5 details;
run;
```

The results are the same as those shown in Output 6.1.1 and Output 6.1.2.

Example 6.3: Running with Data on the SAS appliance

This example uses the same data set as in Example 6.1.

When the input data set resides on the SAS appliance, the SAS appliance performs all computations and reports the results back to the client. In the following example, the input data resides in the MyLib library, which is a distributed data source. To run these statements successfully, you need to set the macro variables to resolve to appropriate values, or you can replace the macro variable references with the appropriate values.

```
option set=GRIDHOST       = "&GRIDHOST";
option set=GRIDINSTALLLOC = "&GRIDINSTALLLOC";

libname MyLib &LIBTYPE
        server  ="&GRIDDATASERVER"
        user    =&USER
        password=&PASSWORD
        database=&DATABASE;

/*MyLib is a libref for a distributed data source
  In this case, the computation is automatically done
  on the SAS appliance.*/
proc hpdmdb data=MyLib.ex print classout=cout varout=vout;
    class x1-x3;
    weight w;
    var x4 y2;
    freq f;
    performance details;
run;
```

The results are the same as those shown in Output 6.1.1 and Output 6.1.2.

Chapter 7
The HPDS2 Procedure

Contents

Overview: HPDS2 Procedure

The HPDS2 procedure enables you to submit DS2 language statements from a Base SAS session to one or more machines in a grid for parallel execution. PROC HPDS2 verifies the syntactic correctness of the DS2 source on the client machine before submitting it for execution. The output data created by the DS2 DATA statement can be output in either of the following ways: it can be written in parallel back to the grid data store or it can be returned to the client machine and directed to any data store that is supported by SAS.

Because the DS2 code is executed in parallel on separate grid nodes that have single data partitions, each node produces separate output that is the result of processing only the local data partition. As a result, it

might be necessary to use a second-stage program to aggregate the results from each node. The second stage can be executed on the SAS client by using the DS2 procedure, where the SET statement reads all rows created by the preceding parallel stage.

The syntax of DS2 is similar to that of the DATA step, but it does not include several key statements such as INPUT and MERGE. In addition, using DS2 along with SAS high-performance analytical procedures limits the PROC DS2 SET statement to a single input stream. The use of BY processing within the SET statement is also not supported. Therefore, many of the traditional DATA step data preparation features are not available in the HPDS2 procedure. PROC HPDS2 is most useful when significant amounts of computationally intensive, row-independent logic must be applied to the data.

For more information about the DS2 language, see *SAS DS2 Language Reference*, which is available at `http://support.sas.com/documentation/solutions/ds2/DS2Ref.pdf`.

PROC HPDS2 runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Server Distributed Mode.

PROC HPDS2 Features

The HPDS2 procedure provides a vehicle for parallel execution of DS2 code in a distributed computing environment. The following list summarizes the basic features of the HPDS2 procedure:

- provides the ability to execute DS2 code in parallel
- enables DS2 code to be executed on a local client machine (single-machine mode) or on multiple machines in a distributed computing environment (distributed mode)
- enables control of the level of parallelism per execution node and the number of nodes to engage
- performs a syntax check of the DS2 code on the local client machine before sending it to the distributed computing environment for execution
- manages data migration to the location of execution and movement back to the client machine as needed

Because the HPDS2 procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Single-Machine and Distributed Execution Modes

The HPDS2 procedure controls both the number of nodes that are engaged and the number of parallel threads that each node uses for the execution of the DS2 language statements. In contrast to the THREADS PACKAGE DS2 (whose syntax provides single-node scalability as part of the DS2 syntax), PROC HPDS2 provides threading that operates outside the syntax of the language.

In single-machine mode, one or more copies of the DS2 program can be executed in parallel in multiple threads on the client machine.

In distributed mode, one or more copies of the DS2 program are executed in parallel on each machine in the distributed computing environment. The distributed mode of execution has two variations:

- In the client-data (local-data) model of distributed execution, the input data are not stored on the appliance but are distibuted to the distributed computing environment by the SAS High-Performance Analytics infrastructure during execution of the HPDS2 procedure.
- In the alongside-the-database model of distributed execution, the data source is the database on the appliance. The data are stored in the distributed database, and the DS2 program that is run on each node is able to read and write the data in parallel during execution of the procedure. Instead of data being moved across the network and possibly back to the client machine, data are passed locally between the processes on each node of the appliance. In general, especially with large data sets, the best HPDS2 performance can be achieved if execution is alongside the database.

By default, the number of copies of the DS2 program that are executed in parallel on a given host (that is, client machine or grid node) is determined by the HPDS2 procedure based on the number of CPUs (cores) available on the host machine. The default is to execute one instance of the DS2 program in a dedicated thread per CPU. You can change the default by specifying the NTHREADS= option in the PERFORMANCE statement. For example, if you specify NTHREADS=*n*, then the HPDS2 procedure runs *n* copies of the DS2 program in parallel on each machine.

For information about the available modes of execution and how to switch between them, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Getting Started: HPDS2 Procedure

This example illustrates a simple HPDS2 procedure. In this case, the DS2 source statements are executed alongside the database in distributed mode. The DS2 code that is submitted to the grid is contained within the DATA and ENDDATA statements. The following DATA step creates a data set that consists of fictitious daily temperatures that are collected from a number of U.S. airports during a period of one week:

```
data daily_temps;
   input city $ mon tue wed thu fri;
datalines;
lax 88 92 94 97 86
sfo 65 60 75 72 74
nyc 99 95 94 95 90
```

```
phl 92 89 91 93 94
atl 95 99 92 98 94
den 85 87 89 72 73
pit 68 70 72 73 77
rdu 98 95 99 95 96
dtt 88 90 90 87 92
anc 51 56 60 64 62
sea 72 78 77 80 79
msy 98 97 99 98 99
mia 95 92 98 94 96
ord 87 85 84 80 79
dfw 95 96 97 95 97
hou 98 99 98 97 92
las 104 105 102 99 101
pdx 78 82 76 74 80
san 80 81 78 82 80
phx 95 98 95 97 100
cle 75 72 76 80 78
ont 78 80 78 81 72
tpa 94 94 92 90 92
bos 80 78 77 75 79
clt 83 80 79 80 81
;
run;
```

The HPDS2 procedure reads this data set and calculates a daily average temperature in Fahrenheit and Celsius for each airport and then provides a synopsis of the weekly temperature average.

In the following statements, the driver DS2GTF.out in the DATA statement refers to the input data set, and the SET DS2GTF.in statement refers to the output data set:

```
libname applianc &ENGINE
        server = "&GRIDDATASERVER"
        user   = &USER
        password = &PASSWORD
        database = &DATABASE;

proc hpds2 data=daily_temps
           out=applianc.avg_temps;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   data DS2GTF.out;
      dcl double avgf avgc;
      dcl char(5) synopsis;
      method run();
         set DS2GTF.in;
         avgf = mean(mon, tue, wed, thu, fri);
         avgc = round((avgf - 32.0) * 5.0/9.0, .1);
         if avgf >= 95.0 then synopsis = 'Hot';
         else if avgf > 80.0 then synopsis = 'Warm';
         else if avgf > 60.0 then synopsis = 'Mild';
         else synopsis = 'Cold';
      end;
   enddata;
run;
```

The following PRINT procedure displays the table of average temperatures that are produced by the HPDS2 procedure:

```
proc print data=applianc.avg_temps;
   title1 'Average Temperatures';
   var city synopsis avgf avgc;
run;
```

Figure 7.1 displays the output of the PRINT procedure.

Figure 7.1 Average Temperatures

Average Temperatures

Obs	city	synopsis	avgf	avgc
1	lax	Warm	91.4	33.0
2	sfo	Mild	69.2	20.7
3	nyc	Warm	94.6	34.8
4	phl	Warm	91.8	33.2
5	atl	Hot	95.6	35.3
6	den	Warm	81.2	27.3
7	pit	Mild	72.0	22.2
8	rdu	Hot	96.6	35.9
9	dtt	Warm	89.4	31.9
10	anc	Cold	58.6	14.8
11	sea	Mild	77.2	25.1
12	msy	Hot	98.2	36.8
13	mia	Hot	95.0	35.0
14	ord	Warm	83.0	28.3
15	dfw	Hot	96.0	35.6
16	hou	Hot	96.8	36.0
17	las	Hot	102.2	39.0
18	pdx	Mild	78.0	25.6
19	san	Warm	80.2	26.8
20	phx	Hot	97.0	36.1
21	cle	Mild	76.2	24.6
22	ont	Mild	77.8	25.4
23	tpa	Warm	92.4	33.6
24	bos	Mild	77.8	25.4
25	clt	Warm	80.6	27.0

Syntax: HPDS2 Procedure

The following statements are available in the HPDS2 procedure:

```
PROC HPDS2 < options > ;
    PERFORMANCE performance-options ;
    DATA DS2GTF.out ;
          DS2 statements
          METHOD RUN()
          SET DS2GTF.in
          END
    ENDDATA ;
    RUN ;
    RUN CANCEL ;
    QUIT ;
```

PROC HPDS2 Statement

PROC HPDS2 < *options* > **;**

The PROC HPDS2 statement invokes the procedure.

You can specify the following options in the PROC HPDS2 statement:

DATA=*SAS-data-set*

IN=*data-set*
names the SAS data set or database table to be used by PROC HPDS2. The default is the most recently created data set.

OUTPUT=*data-set*

OUT=*data-set*
names the SAS data set or database table that is created by PROC HPDS2.

FMTLIBXML=*file-ref*
specifies the file reference for a format stream.

DATA Statement

DATA DS2GTF.out ;

The DATA statement indicates the beginning of the DS2 code block. The code block terminates with the ENDDATA statement.

A reference to the DS2 driver (DS2GTF.out) must be included as part of the DATA statement. If an input data set is specified in the PROC HPDS2 statement, then a **`run()`** method must be included in the DS2 code block and the first statement after the METHOD RUN() statement must be the SET DS2GTF.in statement. DS2GTF.out and DS2GTF.in refer to the input and output data sets, respectively.

ENDDATA Statement

ENDDATA ;

The ENDDATA statement terminates the DS2 code block. The statements between the DATA and ENDDATA statement are submitted to the grid for execution. The DS2 run, init, and term methods are specified between the DATA and ENDDATA statements.

PERFORMANCE Statement

PERFORMANCE *performance-options* **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of PROC HPDS2.

You can also use the PERFORMANCE statement to control whether PROC HPDS2 executes in single-machine or distributed mode.

It is important to remember the distinction between the NODES= and NTHREADS= options. The NODES= option specifies the number of separate distributed nodes that participate in the DS2 execution, whereas the NTHREADS= option determines how many independent copies of the DS2 program are run in parallel on each node. If the data are located on the grid, then all nodes must be engaged; therefore, the NODES= option might be overridden. Setting NODES=0 causes the DS2 code to execute on the client side only. Setting the NTHREADS= option to a value that is greater than the CPU count on each grid node is not likely to improve overall throughput.

For more information about the PERFORMANCE statement, see the section "PERFORMANCE Statement" on page 36 of Chapter 3, "Shared Concepts and Topics."

QUIT Statement

QUIT ;

The QUIT statement stops the procedure. PROC HPDS2 statements that have not been submitted by a RUN statement are terminated.

RUN Statement

RUN ;

The RUN statement submits the PROC HPDS2 statements that precede it for execution.

PROC HPDS2 requires the RUN statement to submit the statements. That is, SAS reads the program statements that are associated with one task until it reaches a RUN statement.

RUN CANCEL Statement

RUN CANCEL ;

The RUN CANCEL statement cancels the PROC HPDS2 statements that precede it. RUN CANCEL is useful if you enter a typographical error.

Details: HPDS2 Procedure

Parallel Execution of DS2 Code

An important characteristic of multithreaded or distributed applications is that they might produce nondeterministic or unpredictable results. The exact behavior of a DS2 program running in parallel on the grid is influenced by a number of factors, including the pattern of data distribution that is used, the execution mode that is chosen, the number of compute nodes and threads that are used, and so on. The HPDS2 procedure does not examine whether the DS2 code that is submitted produces meaningful and reproducible results. It simply executes the DS2 code that is provided on each of the units of work, whether these are multiple threads on a single machine or multiple threads on separate grid nodes. Each instance of the DS2 program operates on a subset of the data. The results that are produced by each unit of work are then gathered, without further aggregation, into the output data set.

Because the DS2 code instances are executed in parallel, consideration must be given to the DS2 language elements that are included in the DS2 code block of an HPDS2 procedure. Not all DS2 language elements can be meaningfully used in multithreaded or distributed applications. For example, lagging or retaining of variables can imply ordering of observations. A deterministic order of observations does not exist in distributed applications, and enforcing data order might have a negative impact on performance.

Optimal performance is achieved when the input data are stored in the distributed database and the grid host is the appliance that houses the data. With the data distributed in this manner, the different instances of the DS2 code running on the grid nodes can read the input data and write the output data in parallel from the local database management system (DBMS).

Limitations and Issues

The current release of the HPDS2 procedure does not support all of the features of the DS2 language. The following subsections summarizes the known limitations and issues for PROC HPDS2.

Packages

DS2 packages are collections of variables and methods that can be used in DS2 programs and threads. The HPDS2 procedure does not support DS2 packages at this time. Use of the PACKAGE and ENDPACKAGE constructs within an HPDS2 procedure results in an error.

PERFORMANCE Statement Options

Setting the NTHREADS= option in the PERFORMANCE statement to very high values can cause out-of-memory errors. For example, out-of-memory errors have been seen with NTHREADS=100.

Data Input/Output

If an input data set is specified, then a SET DS2GTF.in statement must be included in the METHOD RUN() statement. If either the SET DS2GTF.in or the SET DS2GTF.out driver reference is missing, then the SAS session stops responding.

The use of BY groups within the SET statement of an HPDS2 procedure is not supported at this time.

When used within an HPDS2 procedure, the PUT statement does not currently write any data to the client log.

The OVERWRITE= option is not supported in PROC HPDS2.

Data Types and Declarations

The HPDS2 procedure does not support the following data types: REAL, TINYINT, NCHAR, TIMESTAMP, DATE, and TIME. If any of these data types are declared within an HPDS2 procedure, then an error is displayed.

Informats are not currently supported in PROC HPDS2.

Delimited identifiers (for example, dcl double "a%& b") are not currently supported in PROC HPDS2.

No warning or error messages are output when assignments that involve out-of-bounds arrays are used within an HPDS2 procedure.

Error Messages

Incorrect source line numbers are reported when there is an error in an HPDS2 procedure. In addition, the ordering of error messages displayed is reversed for PROC HPDS2 from the order of error messages that is output for DS2.

Examples: HPDS2 Procedure

Example 7.1: Compute Mandelbrot Set

This example computes and plots a Mandelbrot set. The DS2 source statements that compute the set of coordinates that comprise the Mandelbrot set are submitted to the grid and executed alongside the database in distributed mode. Note that Mandelbrot set computation is perfectly scalable in that each point can be computed independently of every other point.

This example uses a DS2 procedure to create a data set that consists of one row for each Mandelbrot coordinate to be computed. The HPDS2 procedure reads this data set and computes the coordinates. The Mandelbrot set is then graphed by using the GCONTOUR procedure.

```
libname applianc &ENGINE
        server = "&GRIDDATASERVER"
        user   = &USER
        password = &PASSWORD
        database = &DATABASE;

/* Set up the table that contains one row for each coordinate to compute */
proc ds2;
   data inp(overwrite=yes);
      dcl double p q r;
      dcl integer maxiterate;
      method init();
         dcl int n m;
         dcl int i j k;
         dcl double pmin pmax qmin qmax;
         n = 1024;
         m = 1024;
         pmin = -1.5; pmax = -0.5;
         qmin = -0.5; qmax =  0.5;
         r = 100.0;
         maxiterate = 50;
         do k = 1 to n*m;
            i = k/m;
            j = mod(k,m);
            p = i*(pmax-pmin)/(n-1)+pmin;
            q = j*(qmax-qmin)/(m-1)+qmin;
            output;
         end;
      end;
   enddata;
run;
quit;

/* Compute the coordinates */
proc hpds2 data=inp out=applianc.mandelbrot;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   data DS2GTF.out;
```

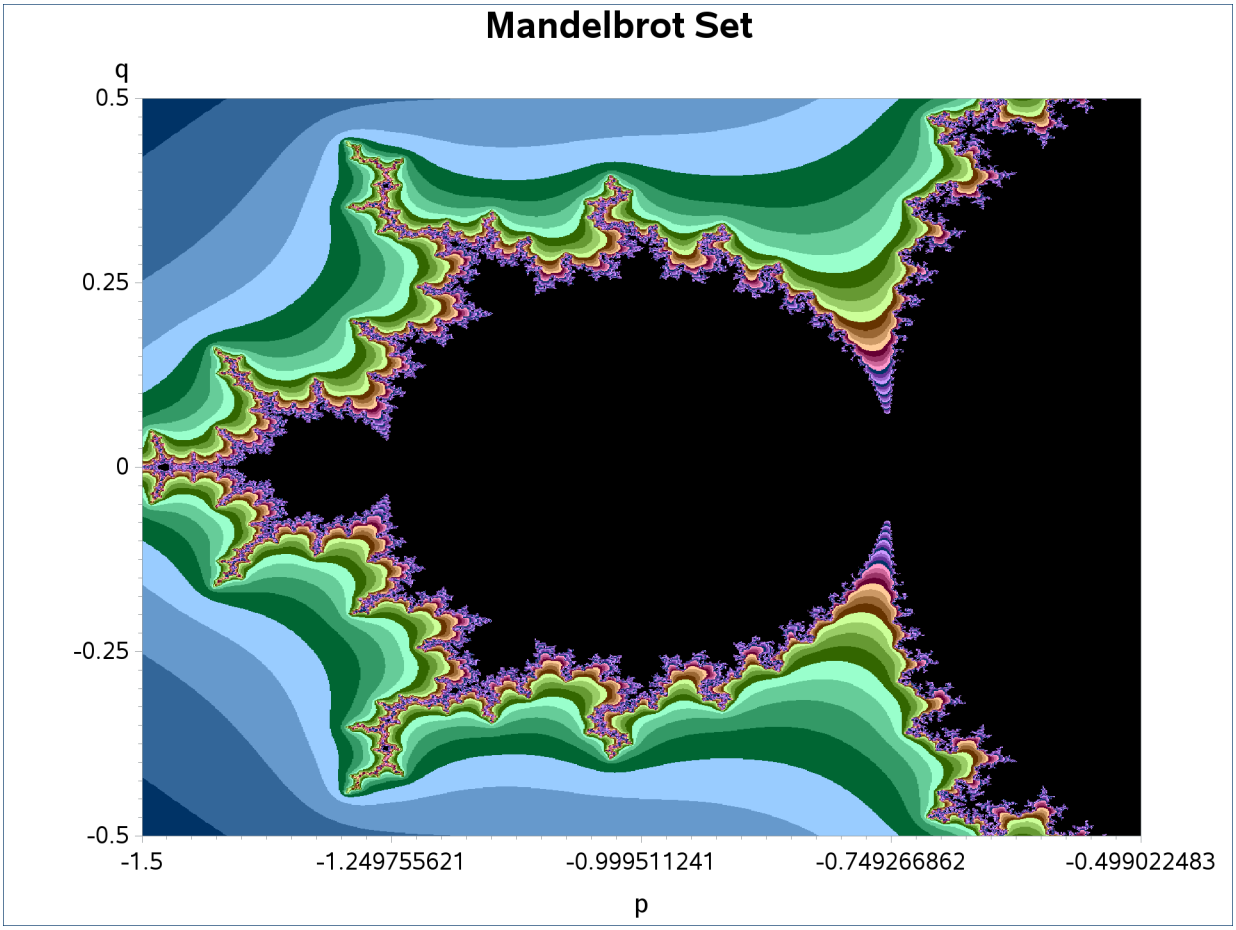
```
      dcl int mesh;
      dcl double x y rr nx ny;
      keep p q mesh;
      method run();
         set DS2GTF.in;
         x = p;
         y = q;
         rr = r**2;
         mesh = 0;
         do while (mesh < maxiterate and (x**2+y**2 < rr));
            nx = x**2 - y**2 + p;
            ny = 2.0*x*y + q;
            x = nx;
            y = ny;
            mesh = mesh+1;
         end;
      end;
   enddata;
run;

/* Plot the results */
goptions colors= (
CX003366 CX336699 CX6699CC CX99CCFF CX006633 CX339966 CX66CC99 CX99FFCC
CX336600 CX669933 CX99CC66 CXCCFF99 CX663300 CX996633 CXCC9966 CXFFCC99
CX660033 CX993366 CXCC6699 CXFF99CC CX003366 CX663399 CX9966CC CXCC99FF
CX003366 CX663399 CX9966CC CXCC99FF CX003366 CX663399 CX9966CC CXCC99FF
CX003366 CX663399 CX9966CC CXCC99FF CX003366 CX663399 CX9966CC CXCC99FF
black
)
;

proc gcontour data=applianc.mandelbrot;
   Title 'Mandelbrot Set';
   plot q*p=mesh /
   nolegend
   pattern
   join
   levels = 5 to 45
  ;
run;
```

Output 7.1.1 shows the graphic representation of the Mandelbrot set that is computed by the HPDS2 procedure.

Output 7.1.1 Computed Mandelbrot Set



Example 7.2: Aggregate Result Data Set

This example illustrates how the intermediate result data that are generated from the DS2 code running in parallel on separate grid nodes can be aggregated into a final result data set. In this case, the aggregation is done by a second-stage PROC DS2 program that executes on the SAS client.

This example uses a DATA step program that runs on the SAS client to generate a sample data set that consists of dimensional information for each of 200 objects (closed cylinders). These data are used by the HPDS2 procedure to calculate the volume and surface area of each object. The second-stage DS2 procedure aggregates these results, summing the total volume and surface area for all objects and computing the average volume and surface area. In this example, the DS2 code running in parallel on the grid is used to perform the row-independent and computationally intensive portion of the processing, whereas the work done by the second-stage DS2 procedure is limited to the final result aggregation and summary.

```
libname applianc &ENGINE
        server = "&GRIDDATASERVER"
        user   = &USER
        password = &PASSWORD
        database = &DATABASE;
```

```
data obj_dims;
   do id=1 to 200;
      radius = ranuni(1) * 10;
      height = ranuni(1) * 20;
      output;
   end;
run;

%let pi=3.14159;
proc hpds2 data=obj_dims
           out=applianc.obj_comps;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   data DS2GTF.out;
      method run();
         set DS2GTF.in;
         volume = &pi * radius**2 * height;
         area = (2 * &pi * radius**2) + (2 * &pi * radius * height);
      end;
   enddata;
run;

proc print data=applianc.obj_comps (obs=20);
   title1 'Volumes and Areas';
run;

data obj_comps;
   set applianc.obj_comps;
run;

proc ds2;
   data obj_totals (keep = (ncount vsum asum vmean amean));
      dcl double ncount vsum asum vmean amean;
      method init();
         ncount = 0;
         vsum = 0;
         asum = 0;
      end;
      method run();
         set {select volume, area from obj_comps};
         ncount + 1;
         vsum + volume;
         asum + area;
      end;
      method term();
         if ncount ne 0 then do;
            vmean = vsum/ncount;
            amean = asum/ncount;
         end;
         output;
      end;
   enddata;
run;
quit;
```

```
proc print data=obj_totals;
   title1 'Total Volume and Area';
run;
```

Output 7.2.1 shows a subset of the volumes and areas that are computed by the HPDS2 procedure.

Output 7.2.1 Computed Volumes and Areas

Volumes and Areas

Obs	id	radius	height	volume	area
1	1	1.8496256982	19.401774313	208.53	246.97
2	2	3.9982430609	5.1879729075	260.55	230.77
3	3	9.2160257787	19.385546995	5172.67	1656.20
4	4	5.4297917315	10.633834456	984.93	548.03
5	5	0.4979402621	1.331331032	1.04	5.72
6	6	8.1931857058	10.477410429	2209.58	961.15
7	7	8.5339431085	1.3436915359	307.43	529.64
8	8	9.5702385761	5.943879283	1710.27	932.89
9	9	2.7261178907	13.798592619	322.16	283.05
10	10	9.7676486241	4.5301503709	1357.82	877.48
11	11	6.8823655028	8.2552773264	1228.45	654.60
12	12	5.5855411271	5.7445122142	563.03	397.63
13	13	4.7578930504	16.89973954	1201.87	647.45
14	14	6.3452411845	11.807293385	1493.47	723.71
15	15	5.8258152641	7.5402673835	803.99	489.26
16	16	7.2836155991	10.132070589	1688.66	797.02
17	17	9.3121359401	18.582400996	5062.32	1632.10
18	18	5.8966033794	5.9444569265	649.33	438.70
19	19	3.910424334	9.4485835123	453.90	328.23
20	20	6.7952574821	3.3617670198	487.67	433.66

Output 7.2.2 shows the aggregated results that are produced by the second-stage DS2 program.

Output 7.2.2 Computed Total Volume and Area

Total Volume and Area

Obs	ncount	vsum	asum	vmean	amean
1	200	209883.99	104680.26	1049.42	523.401

Chapter 8
The HPIMPUTE Procedure

Contents

Overview: HPIMPUTE Procedure

The HPIMPUTE procedure executes high-performance numeric variable imputation. Imputation is a common step in the data preparation stage. The HPIMPUTE procedure takes only numeric variables.

The HPIMPUTE procedure can replace numeric missing values with a specified value. It can also replace numeric missing values with the mean, the pseudomedian, or some random value between the minimum value and the maximum value of the nonmissing values.

When PROC HPIMPUTE calculates the mean, the pseudomedian, or a random value, it ignores any observation that has a FREQ variable whose value is less than or equal to 0.

The HPIMPUTE procedure runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Server Distributed Mode.

PROC HPIMPUTE Features

PROC HPIMPUTE provides a vehicle for the parallel execution of imputation. The following list summarizes the basic features of PROC HPIMPUTE:

- provides the ability to execute imputation
- manages data migration to the location of execution and movement back to the client machine as needed

Because the HPIMPUTE procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Single-Machine and Distributed Execution Modes

The HPIMPUTE procedure can exploit computer grids by imputing independently on different grid nodes in parallel, and it supports multithreading on each node. For more information about single-machine and distributed execution modes, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

You can control both the number of parallel threads per execution node and the number of computing nodes to engage.

Alternatively, PROC HPIMPUTE can be executed on a grid of distributed computers. In distributed mode, one or more copies of the imputation code are executed in parallel on each grid node.

The distributed mode of execution has two variations:

- In the client-data (local-data) model of distributed execution, the input data are not stored on the appliance but are distributed to the distributed computing environment during execution of the HPIMPUTE procedure.
- In the alongside-the-database model of distributed execution, the data source is the database on the appliance. The data are stored in the distributed database, and the imputation code that runs on each node can read and write the data in parallel during execution of the HPIMPUTE procedure. Instead of being moved across the network and possibly back to the client machine, data are passed locally between the processes on each node of the appliance. In general and especially with large data sets, the best PROC HPIMPUTE performance can be achieved if execution is alongside the database.

Getting Started: HPIMPUTE Procedure

The HPIMPUTE procedure can use four methods to impute numeric missing values. This example uses all four imputation methods available in the IMPUTE statement to manipulate a data set. The following SAS DATA step creates the SAS data set ex1, which has six variables: the first four variables all have some missing values, the fifth variable is the frequency variable, and the last variable is an index variable.

```
data ex1;
input a  b  c  d  freq id;
cards;
       2    3   1  1   2   1
       2    2   2  2   3   2
       .    0   3  .   0   3
       2    3   .  .   .   4
       2    .   .  .   -5  5
       .    6   .  .   3   6
       .    4   .  .   4   7
       2    5   .  .   3   8
       .    6   9  9   1   9
       2    3   10 10  3   10
run;
```

The following statements include four IMPUTE statements, each of which specifies a different imputation method. The INPUT statement specifies the input variables. PROC HPIMPUTE assumes that the variables have an interval level of measurement because the variables are numeric.

```
 proc hpimpute data=ex1 out=out1;
     id id;
     input a b c d;
     impute a / value=0.1;
     impute b / method=pmedian;
     impute c / method=random;
     impute d / method=mean;
     freq freq;
 run;
```

Output 8.1 shows the imputation results. The Variable column shows the original variable names from the input data set. The Imputation Indicator column shows a 0 if that observation was not imputed and 1 if it was. The Imputed Variable column shows the names of imputed variables in the output data set. The Type of Imputation column shows the types of imputation methods: Given Value, Pseudo Median, Random (between the minimum value and the maximum value of the nonmissing values), and Mean. For random imputation, the last column shows the imputation seed. For other imputation methods, the last column shows the imputation value that is used to replace missing values.

Figure 8.1 HPIMPUTE Getting Started Example Output

The HPIMPUTE Procedure

Imputation Results

Variable	Imputation Indicator	Imputed Variable	N Missing	Type of Imputation	Imputation Value (Seed)
a	M_a	IM_a	4	Given value	0.10000
b	M_b	IM_b	1	Pseudo Median	4.00000
c	M_c	IM_c	5	Random	5.00000
d	M_d	IM_d	6	Mean	5.22222

Syntax: HPIMPUTE Procedure

The following statements are available in the HPIMPUTE procedure:

PROC HPIMPUTE < *options* > **;**
INPUT *variables* **;**
IMPUTE *variables* < *options* > **;**
PERFORMANCE *performance-options* **;**
ID *variables* **;**
FREQ *variable* **;**
CODE < *options* > **;**

The PROC HPIMPUTE statement, INPUT statement, and IMPUTE statement are required. The INPUT statement and IMPUTE statement can appear multiple times.

PROC HPIMPUTE Statement

PROC HPIMPUTE < *options* > **;**

The PROC HPIMPUTE statement invokes the procedure. You can specify one or both of the following *options*:

DATA=< *libref.* **>***SAS-data-set*
names the SAS data set for which PROC HPIMPUTE is to impute values. The default is the most recently created data set. If the data are already distributed, PROC HPIMPUTE reads the data alongside the distributed database. For information about the various execution modes and about the alongside-the-database model, see the sections "Processing Modes" on page 10 and "Alongside-the-Database Execution" on page 18 in Chapter 3, "Shared Concepts and Topics.". WHERE processing is supported.

OUT=< *libref.* **>***SAS-data-set*
names the SAS data set to which PROC HPIMPUTE writes the data along with ID variables (if applicable), imputation indicator variables (0 for not imputed or 1 for imputed), and imputed variables. There is no default output data set.

CODE Statement

CODE < *options* > **;**

The CODE statement generates SAS DATA step code that mimics the computations that are performed when the IMPUTE statement runs in single-machine mode and uses a single thread. You can specify the following *options*:

FILE=*filename*
specifies the name of the file to which the SAS score code is written.

FREQ Statement

FREQ *variable* **;**

The *variable* in the FREQ statement identifies a numeric variable in the data set that contains the frequency of occurrence for each observation. PROC HPIMPUTE treats each observation as if it appeared n times, where n is the value of the FREQ variable for the observation. If the frequency value is not an integer, it is truncated to an integer. If the frequency value is less than 1 or missing, the observation is not used in the analysis. When the FREQ statement is not specified, each observation is assigned a frequency of 1.

ID Statement

ID *variables* **;**

The optional ID statement lists one or more *variables* from the input data set that are transferred to the output data set. The ID statement accepts numeric and character variables. For example, when you specify the OUT= option in the PROC HPIMPUTE statement, the ID variables, followed by indicator variables and imputed variables, are added to the output data set.

By default, to avoid data duplication for large data sets, the HPIMPUTE procedure does not include any variables from the input data set in the output data sets. Therefore, the ID statement can be used to copy variables from the input data set to the output data set.

The variables in an ID statement must not appear in any INPUT statement; if they appear, an error is reported.

IMPUTE Statement

IMPUTE *variables* < *options* > **;**

The IMPUTE statement names the variables whose values PROC HPIMPUTE imputes. You can specify multiple IMPUTE statements. The IMPUTE statement takes only numeric variables; character variables are reported as an error. The variables in an IMPUTE statement must appear in an INPUT statement; if they do not appear in an INPUT statement, an error is reported.

You can specify the following *options*:

METHOD=MEAN | RANDOM | PMEDIAN
specifies the method of imputation. You can specify the following values:

MEAN	replaces missing values with the algebraic mean of the variable. If there is no nonmissing value for the variable to be imputed, the imputation result remains missing.
RANDOM	replaces missing values with a random value that is drawn between the minimum and maximum of the variable. If there is no nonmissing value for the variable to be imputed, the imputation result remains missing.
PMEDIAN	replaces missing values with the pseudomedian of the variable. If there is no nonmissing value for the variable to be imputed, the imputation result remains missing.

NOTE: If you specify the method of imputation and all the values for all the variables to be imputed are missing, PROC HPIMPUTE exits with an error.

VALUE=*value*
replaces missing values with the specified *value*.

INPUT Statement

INPUT *variables* **;**

The INPUT statement names one or more input variables. You can specify multiple INPUT statements. The INPUT statement takes only numeric variables; character variables are reported as an error.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of the HPIMPUTE procedure.

You can also use the PERFORMANCE statement to control whether the HPIMPUTE procedure executes in single-machine or distributed mode.

The PERFORMANCE statement is documented further in the section "PERFORMANCE Statement" on page 36 of Chapter 3, "Shared Concepts and Topics."

Details: HPIMPUTE Procedure

Obtaining the Statistics for Imputation

PROC HPIMPUTE first computes the imputation value and then imputes with that value. Some statistics (such as the mean, minimum, and maximum) are computed precisely. The pseudomedian, which is calculated if you specify METHOD=PMEDIAN in the IMPUTE statement, is an estimation of the median. The computation of the median requires sorting the entire data. However, in a distributed computing environment, each grid node contains only a part of the entire data. Sorting all the data in such an environment requires a lot of internode communications, degrading the performance dramatically. To address this challenge, a binning-based method is used to estimate the pseudomedian.

For variable x, assume that the data set is $\{x_i\}$, where $i = 1, 2, ..., n$. Let $min(x) = \min_{i \in \{1..n\}} \{x_i\}$, and let $max(x) = \max_{i \in \{1..n\}} \{x_i\}$. The range of the variable is $range(x) = max(x) - min(x)$.

A simple bucket binning method is used to obtain the basic information. Let N be the number of buckets, ranging from $min(x)$ to $max(x)$. For each bucket B_i, $i = 1, 2, ..., N$, PROC HPIMPUTE keeps following information:

- c_i: count of x in B_i
- min_i: minimum value of x in B_i
- max_i: maximum value of x in B_i

For each bucket B_i, the range R_i is

$$R_i = \begin{cases} [min(x) + (i-1) * d, min(x) + i * d) & \text{if } i < N \\ [min(x) + (i-1) * d, max(x)] & \text{if } i = N \end{cases}$$

where $d = \frac{max(x) - min(x)}{N}$

To calculate the pseudomedian, PROC HPIMPUTE finds the smallest I, such that $\sum_{i=1}^{I} c_i >= 0.5 * m$, where m is the number of nonmissing observations of x in the data set. Therefore, the pseudomedian value Q is

$$Q = \begin{cases} min_I & \text{if } \sum_{i=1}^{I} c_i > 0.5 * m \\ max_I & \text{if } \sum_{i=1}^{I} c_i = 0.5 * m \end{cases}$$

N is set to 10,000 in PROC HPIMPUTE. Experiments show that the pseudomedian is a good estimate of the median and that the performance is satisfactory.

Random Imputation

If you specify METHOD=RANDOM in the IMPUTE statement, PROC HPIMPUTE replaces missing values with a random value that is drawn between the minimum and maximum of the variable. For

variable x, assume that the data set is $\{x_i\}$, where $i = 1, 2, ..., n$. Let $min(x) = \min_{i \in \{1..n\}} \{x_i\}$, and let $max(x) = \max_{i \in \{1..n\}} \{x_i\}$. The random value $R = min(x) + (max(x) - min(x)) * ranuni(SEED)$, where $ranuni()$ is a function that takes a SEED (number) as input and returns a random value from a uniform distribution between 0 and 1. When PROC IMPUTE runs in single-machine mode and uses a single thread, the SEED is set to 5. When PROC IMPUTE runs in distributed execution mode or uses multiple threads, the SEED is determined at run time.

Displayed Output

The HPIMPUTE procedure displays imputation results and performance information.

The "Imputation Results" table includes six columns. The first column shows the original variable names from the input data set. The second column displays a 0 if that observation was not imputed and a 1 if it was. The third column shows the names of imputed variables in the output data set. The fourth column shows the number of missing values. The fifth column shows the types of imputation methods: Given Value, Mean, Pseudo Median, Random (between the minimum value and the maximum value of the nonmissing values). The last column shows the imputation seed for which random imputation generated the imputation value, or the imputation value for other imputation methods that replace missing values.

The "Performance Information" table is produced by default. It displays information about the execution mode. For single-machine mode, the table displays the number of threads used. For distributed mode, the table displays the grid mode (symmetric or asymmetric), the number of compute nodes, and the number of threads per node.

If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a "Timing" table in which elapsed times (absolute and relative) for the main tasks of the procedure are displayed.

ODS Table Names

Table 8.1 lists the names of the data tables that are created by the HPIMPUTE procedure. You must use these names in ODS statements.

Table 8.1 ODS Tables Produced by PROC HPIMPUTE

Table Name	Description	Required Statement or Option
PerformanceInfo	Performance information	Default output
ImputeResults	Imputation results	Default output
Timing	Timing	PERFORMANCE statement with DETAILS option

Examples: HPIMPUTE Procedure

Example 8.1: Running Client Data on the Client

This example demonstrates how to use PROC HPIMPUTE to perform imputation on the Sampsio.Hmeq data set, which resides on the client.

When the input data set resides on the client and no PERFORMANCE statement is specified, as in the following statements, the client performs all computations:

```
/*sampsio is a libref for a data source on the client.*/
proc hpimpute data=sampsio.hmeq out=out1;
    input mortdue value clage debtinc;
    impute mortdue / value  = 70000;
    impute value   / method = mean;
    impute clage   / method = random;
    impute debtinc / method = pmedian;
run;
```

Output 8.1.1 shows the imputation results.

Output 8.1.1 Imputation Results

The HPIMPUTE Procedure

Imputation Results

Variable	Imputation Indicator	Imputed Variable	N Missing	Type of Imputation	Imputation Value (Seed)
MORTDUE	M_MORTDUE	IM_MORTDUE	518	Given value	70000
VALUE	M_VALUE	IM_VALUE	112	Mean	101776
CLAGE	M_CLAGE	IM_CLAGE	308	Random	5.00000
DEBTINC	M_DEBTINC	IM_DEBTINC	1267	Pseudo Median	34.81696

Output 8.1.2 shows the performance information.

Output 8.1.2 Performance Information

The HPIMPUTE Procedure

Performance Information

Execution Mode	Single-Machine
Number of Threads	4

Example 8.2: Running Client Data on the SAS Appliance

This example uses the same data set as in Example 8.1.

When the input data set resides on the client and a PERFORMANCE statement that includes a NODES= option is specified, as in the following statements, PROC HPIMPUTE copies the data set to the SAS appliance,

where the imputation is performed:

```
/*Perform the computation on the SAS appliance using 2 nodes*/
option set=GRIDHOST="&GRIDHOST";
option set=GRIDINSTALLLOC="&GRIDINSTALLLOC";
proc hpimpute data=sampsio.hmeq out=out2;
    input mortdue value clage debtinc;
    impute mortdue / value  = 70000;
    impute value   / method = mean;
    impute clage   / method = random;
    impute debtinc / method = pmedian;
    performance nodes=2 details
    host="&GRIDHOST" install="&GRIDINSTALLLOC";
run;
```

Output 8.2.1 shows the imputation results.

Output 8.2.1 Imputation Results

The HPIMPUTE Procedure

Imputation Results

Variable	Imputation Indicator	Imputed Variable	N Missing	Type of Imputation	Imputation Value (Seed)
MORTDUE	M_MORTDUE	IM_MORTDUE	518	Given value	70000
VALUE	M_VALUE	IM_VALUE	112	Mean	101776
CLAGE	M_CLAGE	IM_CLAGE	308	Random	5.00000
DEBTINC	M_DEBTINC	IM_DEBTINC	1267	Pseudo Median	34.81696

Output 8.2.2 shows the performance information.

Output 8.2.2 Performance Information

Performance Information

Host Node	<< your grid host >>
Install Location	<< your grid install location >>
Execution Mode	Distributed
Number of Compute Nodes	2
Number of Threads per Node	32

Output 8.2.3 shows the timing information.

Output 8.2.3 Procedure Task Timing

Procedure Task Timing

Task	Seconds	Percent
Startup of Distributed Environment	2.80	96.79%
Data Transfer from Client	0.03	1.01%
Computation	0.02	0.55%
Writing Output	0.05	1.65%

Chapter 9
The HPSAMPLE Procedure

Contents

Overview: HPSAMPLE Procedure

The HPSAMPLE procedure is a high-performance procedure that performs either simple random sampling or stratified sampling. The HPSAMPLE procedure runs in either single-machine mode or distributed mode.
NOTE: Distributed mode requires SAS High-Performance Server Distributed Mode.

The HPSAMPLE procedure creates the following:

- one output data set, which contains the sample data set
- one performance table, which contains performance information
- one frequency table, which contains the frequency information for the population and sample

Because the HPSAMPLE procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations

- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode (currently HPSAMPLE is only using a single thread)

For more information, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Getting Started: HPSAMPLE Procedure

The following example shows a 10% stratified sampling, with the target variable BAD used by the HPSAMPLE procedure as a stratum:

```
proc hpsample data=Sampsio.Hmeq  out=Smp samppct=10 seed=1234 partition;
    var loan derog mortdue value yoj delinq
        clage ninq clno debtinc;
    class bad reason job;
    target bad;
run;
proc print data=Smp;run;
```

The input data set Sampsio.Hmeq includes information about 5,960 fictitious mortgages. Each observation represents an applicant for a home equity loan, and all applicants have an existing mortgage. The SAMPPCT=10 option specifies that 10% of the input data be sampled. The SEED option specifies that the random seed used in the sampling process be 1234. The PARTITION option specifies that the output data set, Smp, include an indicator that shows whether each observation is selected to the sample (1) or not (0). The VAR statement specifies 10 numeric input variables, and the CLASS statement specifies three classification input variables. All these variables are included in the output sample. The binary TARGET variable BAD indicates whether an applicant eventually defaulted or was ever seriously delinquent. The TARGET statement triggers stratified sampling, which enables you to sample each subpopulation in the target variable (stratum) independently. The displayed output contains a performance table (Figure 9.1) that shows the performance environment information and a frequency table (Figure 9.2) that shows the frequency of observations in each level of BAD.

Figure 9.1 Performance Information

The HPSAMPLE Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	1

Figure 9.2 Frequency Table

One Target Stratified Sampling Frequency Table		
Target Level	**Number of Obs**	**Number of Samples**
0	4771	478
1	1189	118

Syntax: HPSAMPLE Procedure

The following statements are available in the HPSAMPLE procedure:

PROC HPSAMPLE < *options* > **;**
VAR *variable* < *variable ... variable* > **;**
CLASS *variable* < *variable ... variable* > **;**
TARGET *variable* < *variable ... variable* > **;**
PERFORMANCE *performance-options* **;**

Either a VAR or a CLASS statement is required for simple random sampling; both the CLASS and TARGET statements are required for stratified sampling.

PROC HPSAMPLE statement

PROC HPSAMPLE < *options* > **;**

The PROC HPSAMPLE statement invokes the procedure.

You can specify the following *options*:

DATA=< *libref.* **>***table*
names the table (SAS data set or database table) that you want to sample from. The default is the most recently opened or created data set. If the data are already distributed, the procedure reads the data alongside the distributed database. See the section "Single-machine Mode and Distributed Mode" on page 10 for the various execution modes and the section "Alongside-the-Database Execution" on page 15 for the alongside-the-database model.

NONORM
distinguishes target values that share the same normalized value when you perform stratified sampling or oversampling. For example, if a target has three distinct values, "A", "B", and "b", and you want to treat "B" and "b" as different levels, you need to use NONORM. By default, "B" and "b" are treated as the same level. PROC HPSAMPLE normalizes a value as follows:

1. Leading blanks are removed.
2. The value is truncated to 32 characters.
3. Letters are changed from lowercase to uppercase.

Note: In the oversampling case, there is no normalization for levels by default. If you do not specify this option, you need to specify a normalized event value in the EVENT= option.

OUT=< *libref.* **>***SAS-data-set*
names the SAS data set that you want to output the sample to. If you run alongside the database, you need to specify a data set that has the same database *libref* as the input data set and make sure it does not already exist in the database. This option is required.

Note: This SAS data set will contain the sample data set, which includes variables that are specified in VAR and CLASS statements. If you also specify the PARTITION option, the output includes one

more column, _PartInd_. In the oversampling case, an additional column, _Freq_, is provided. It is calculated as the ratio of rare level's proportion in the population to its proportion in the sample.

PARTITION
produces an output data set that has the same number of rows as the input data set but has an additional partition indicator (_PARTIND_), which indicates whether an observation is selected to the sample (1) or not (0). If you also specify the SAMPPCT2= option, _PARTIND_ indicates whether an observation is selected to the sample1 (1), the sample2 (2), or the rest (0).

PARTINDNAME=*partition-indicator-name*
renames the partition indicator (_PARTIND_) to the specified *partition-indicator-name*.

SEED=*random-seed*
specifies the seed for the random number generator. If you do not specify *random-seed* or you specify it as a nonpositive number, the seed is set to be the default 12345. The SEED option enables you to reproduce the same sample output.

You can specify the following *options* only for simple random sampling and stratified sampling:

SAMPOBS=*number*
specifies the minimum number of observations you want to sample from the input data. The value of *number* must be a positive integer. If *number* exceeds the total number of observations in the input data, the output sample has the same number of observations as the input data set.

SAMPPCT=*sample-percentage*
specifies the sample percentage to be used by PROC HPSAMPLE. The value of *sample-percentage* should be a positive number less than 100. For example, SAMPPCT=50.5 specifies that you want to sample 50.5% of data.

NOTE: You must specify either the SAMPOBS or the SAMPPCT option if you want to perform simple random sampling or stratified sampling. If you specify both options, only the SAMPPCT option is honored.

SAMPPCT2=*sample-percentage*
partitions the input data into three parts when specified along with the SAMPPCT= and PARTITION options. The percentage of the sample whose _PARTIND_=1 is specified in the SAMPPCT= option, the percentage of the sample whose _PARTIND_=2 is specified in the SAMPPCT2= option, and the percentage of the sample whose _PARTIND_=0 is 100 minus the sum of the values of the SAMPPCT= and SAMPPCT2= options. The sum of the *sample-percentages* specified in this option and in the SAMPPCT2= option must be a positive number less than 100.

You can specify the following *options* only for oversampling:

EVENT=*"rare-event-level"*
specifies the rare event level. If you specify this option, PROC HPSAMPLE uses an oversampling technique to adjust the class distribution of a data set, and the following two options are required.

SAMPPCTEVT=*sample-event-percentage*
specifies the sample percentage from the event level. The value of *sample-event-percentage* should be a positive number less than 100. For example, SAMPPCTEVT=50.5 specifies that you want to sample 50.5 percent of the rare event level.

EVENTPROP=*event-proportion*
specifies the proportion of rare events in the sample. The value of *event-proportion* should be a positive number less than 1. For example, EVENTPROP=0.3 specifies that you want the ratio between rare events and not rare events to be 3:7.

CLASS statement

CLASS *variable* < *variable ... variable* > **;**

The CLASS statement specifies one or more classification variables to be included in the sample. At least one *variable* is required. A *variable* can be character or numeric. The CLASS and VAR statements are mutually exclusive.

NOTE: Each *variable* in the TARGET statement must be specified in the CLASS statement. And the order of Target variables in the output frequency table is the same as the order of these varibles specified in CLASS statement.

PERFORMANCE statement

PERFORMANCE < *performance-options* > **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables about the distributed computing environment, and requests detailed results about the performance characteristics of a SAS high-performance analytical procedure.

You can also use the PERFORMANCE statement to control whether a SAS high-performance analytical procedure executes in single-machine mode or distributed mode.

The PERFORMANCE statement for SAS high-performance analytical procedures is documented in the section "PERFORMANCE Statement" on page 36 of Chapter 3, "Shared Concepts and Topics."

NOTE: PROC HPSAMPLE does not support multithreading in this release.

TARGET statement

TARGET *variable* < *variable ... variable* > **;**

The TARGET statement specifies classification variables to be used for stratification. Each *variable* must be specified in the CLASS statement. Currently, up to four target variables are supported for stratified sampling, and one target variable is supported for oversampling. The maximum number of levels (that is, distinct values) in any target variable is 256.

VAR statement

VAR *variable* < *variable ... variable* > **;**

The VAR statement specifies one or more numeric variables to be included in the sample. At least one *variable* is required; all *variables* must be numeric. You can use this statement to include only the variables of interests in your sample. The CLASS and VAR statements are mutually exclusive.

Details: HPSAMPLE Procedure

Class Level

For classification variables, a *level* is an observed value that is distinct after formatting, removal of beginning and ending white space, and capitalization. For example, the values **`MyLevel`** and **`MYLEVEL`** are treated as a single level in the data set. Class variables can be numeric, and the same levelization rules apply. For example, **`3.000002`** and **`3.0000001`** are treated as the same level if they are formatted using **`BEST3`**.

Displayed Output

The following sections describe the output that PROC HPSAMPLE produces by default. The output is organized into various tables, which are discussed in the order of their appearance.

Performance Information

The "Performance Information" table is produced by default. It displays information about the execution mode. For single-machine mode, the table displays the number of threads used. For distributed mode, the table displays the grid mode (symmetric or asymmetric), the number of compute nodes, and the number of threads per node.

If you specify the DETAILS option in the PERFORMANCE statement, the procedure also produces a "Timing" table in which elapsed times (absolute and relative) for the main tasks of the procedure are displayed.

Timing Table

The "Timing Table" lists the timing information for various computational stages of the procedure.

Frequency Information Table

For simple random sampling, the "Frequency Information Table" lists the number of observations in the input data set and in the sample output data set.

For stratified sampling, the "Frequency Information Table" table lists the respective frequency in each stratum for the input data and the sample. If one target variable is specified, each level of the target variable represents a stratum; if two target variables are specified, a combination of the levels of two target variables represents a stratum.

ODS Table Names

Each table that the HPSAMPLE procedure creates has a name associated with it. You must refer to the table by this name when you use ODS statements. These names are listed in Table 9.1.

Table 9.1 ODS Tables Produced by PROC HPSAMPLE

Table Name	Description	Required Statement / Option
PerformanceInfo	Information about the high-performance computing environment	Default output
Timing	Timing information for various computational stages of the procedure	DETAILS (PERFORMANCE statement)
FreqTable	Frequency table of input data set and output sample (when target variables are used, this table contains stratification information for population and sample)	Default output

Examples: HPSAMPLE Procedure

Example 9.1: Running PROC HPSAMPLE on the Client

This example demonstrates how to use PROC HPSAMPLE to perform simple random sampling on the Sampsio.Hmeq data set, which resides on the client.

When the input data set resides on the client and no PERFORMANCE statement is specified, as in the following statements, the client performs all computations:

```
/*sampsio is a libref for a data source on the client.*/
proc hpsample data=sampsio.hmeq out=out1 sampobs=20 seed=13579;
    class job reason;
    var loan value delinq derog;
run;
proc print data=out1;
run;
```

Output 9.1.1 shows the performance environment information.

Output 9.1.1 Performance Information

The HPSAMPLE Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	1

Output 9.1.2 shows the number of observations in the data set sampsio.hmeq and the number of samples.

Output 9.1.2 Frequency Table

Simple Random Sampling
Frequency Table

Number of Obs	Number of Samples
5960	22

Output 9.1.3 shows the sample data.

Output 9.1.3 Sample Data

Obs	JOB	REASON	LOAN	VALUE	DELINQ	DEROG
1	Other	HomeImp	4900	65774	2	1
2	Other	HomeImp	5700	82923	0	0
3	Other	HomeImp	7000	124827	.	.
4		DebtCon	8300	75081	0	0
5	Office	DebtCon	10000	125500	0	0
6	Other		11100	61406	0	0
7	Other	HomeImp	11500	64037	0	0
8	ProfExe	HomeImp	11900	105454	0	0
9	Mgr	HomeImp	15000	122400	2	0
10	Other	HomeImp	15000	107207	0	0
11	ProfExe	HomeImp	15600	106824	0	0
12	Office	DebtCon	17000	69000	0	0
13	ProfExe	DebtCon	17300	49100	0	0
14	Other	DebtCon	18000	60000	0	2
15	Office	DebtCon	21100	98000	2	0
16	Other	HomeImp	21400	103427	0	0
17	Mgr	HomeImp	22400	121601	0	0
18	Other	HomeImp	25000	202500	0	0
19	Other	DebtCon	25500	43031	.	.
20	Mgr	DebtCon	27500	149877	0	0
21	ProfExe	DebtCon	36500	195729	0	0
22	Self	DebtCon	70300	294169	0	0

Example 9.2: Running with Client Data on the SAS Appliance

This example uses the same data set as is used in Example 9.1. This example demonstrates how to use PROC HPSAMPLE to perform stratified sampling.

When the input data set resides on the client and a PERFORMANCE statement with a NODES= option is specified, as in the following statements, PROC HPSAMPLE copies the data set to the SAS appliance, where the sampling is performed:

```
/*Perform the computation on the SAS appliance using 2 nodes*/
option set=GRIDHOST="&GRIDHOST";
option set=GRIDINSTALLLOC="&GRIDINSTALLLOC";
proc hpsample data=sampsio.hmeq out=out2 samppct=10 seed=13579 partition;
    var loan value delinq derog;
    class job reason;
    target job;
    performance nodes = 2;
run;
 proc print data=out2(obs=15);
 run;
```

Output 9.2.1 shows the performance environment information.

Output 9.2.1 Performance Information

The HPSAMPLE Procedure

Performance Information	
Host Node	bigmath.unx.sas.com
Execution Mode	Distributed
Number of Compute Nodes	2
Number of Threads per Node	1

Output 9.2.2 shows the frequency information for each level of target variable JOB in the data set Sampsio.Hmeq and in the sample.

Output 9.2.2 Frequency Table

One Target Stratified Sampling Frequency Table		
Target Level	**Number of Obs**	**Number of Samples**
	279	28
MGR	767	77
OFFICE	948	95
OTHER	2388	239
PROFEXE	1276	127
SALES	109	11
SELF	193	19

Output 9.2.3 shows the first 15 output sample observations that contain "_PARTIND_", which indicates whether the observation is selected for the sample (1) or not (0).

Output 9.2.3 Sample Output with Partition Indicator

Obs	JOB	REASON	LOAN	VALUE	DELINQ	DEROG	_PartInd_
1	Other	HomeImp	1100	39025	0	0	0
2	Other	HomeImp	1300	68400	2	0	0
3	Other	HomeImp	1500	16700	0	0	0
4			1500	.	.	.	0
5	Office	HomeImp	1700	112000	0	0	0
6	Other	HomeImp	1700	40320	0	0	0
7	Other	HomeImp	1800	57037	2	3	0
8	Other	HomeImp	1800	43034	0	0	1
9	Other	HomeImp	2000	46740	2	0	0
10	Sales	HomeImp	2000	62250	0	0	0
11			2000	.	.	.	0
12	Office	HomeImp	2000	29800	1	0	1
13	Other	HomeImp	2000	55000	0	0	0
14	Mgr		2000	87400	0	0	0
15	Other	HomeImp	2100	83850	1	0	0

Example 9.3: Running with Data on the SAS Appliance

This example uses the same data set as in Example 9.1. It demonstrates how to use PROC HPSAMPLE to perform oversampling.

When the input data set resides on the SAS appliance, the SAS appliance performs all samplings, writes out a sample on the SAS appliance, and reports the frequency results back to the client. In the following statements, the input data resides in the MyLib library (which is a distributed data source), and the output data set is a distributed data set in the MyLib library. You can use PROC DATASETS to delete the output table if it already exists on the SAS appliance. The **ods output FreqTable=Freqtab;** statement saves the frequency table to a SAS data set called Freqtab on the client.

```
/*MyLib is a libref for a distributed data source
  In this case, the computation is automatically done
  on the SAS Appliance.*/

option set=GRIDHOST       = "&GRIDHOST";
option set=GRIDINSTALLLOC = "&GRIDINSTALLLOC";
option set=GRIDMODE = "&GRIDMODE";
libname MyLib &LIBTYPE
        server  ="&GRIDDATASERVER"
        user    =&USER
        password=&PASSWORD
        database=&DATABASE;

proc delete data=MyLib.out_smp_ex3; run;

proc hpsample data=MyLib.hmeq out=MyLib.out_smp_ex3 seed=13579 partition
   samppctevt=80  eventprop=.2 event="SALES";
    var loan value delinq derog;
    class job;
    target job;
```

```
    ods output FreqTable=Freqtab;
run;
```

Output 9.3.1 shows the performance environment information.

Output 9.3.1 Performance Information

The HPSAMPLE Procedure

Performance Information	
Host Node	greenarrow.unx.sas.com
Execution Mode	Distributed
Number of Compute Nodes	16
Number of Threads per Node	1

Output 9.3.2 shows the number of observations in each level of target variable JOB in the data set MyLib.Hmeq and in the sample. After oversampling, the proportion of SALES level is adjusted to 20% in the sample from the original 1.8% in the population.

Output 9.3.2 Frequency Table

Oversampling Frequency Table		
Target Level	**Number of Obs**	**Number of Samples**
	279	15
MGR	767	47
OFFICE	948	56
OTHER	2388	142
PROFEXE	1276	77
SALES	109	87
SELF	193	15

Chapter 10
The HPSUMMARY Procedure

Contents

Overview: HPSUMMARY Procedure

The HPSUMMARY procedure computes basic descriptive statistics for variables in a SAS data set. It is a high-performance version of the SUMMARY procedure in Base SAS. PROC HPSUMMARY runs in either single-machine mode or distributed mode.

NOTE: Distributed mode requires SAS High-Performance Server Distributed Mode.

When run in distributed mode, the HPSUMMARY procedure enables you to summarize data that have been distributed to the grid for parallel execution. The output data that PROC HPSUMMARY creates can then be written in parallel back to the grid data store.

In distributed mode, it is recommended that input data reside on the grid and that results be output back to the grid. Although it is possible to use PROC HPSUMMARY on data that do not reside on the grid or to produce result tables that do not reside on the grid, this usage is not recommended because of the overhead of transferring data to and from the grid.

PROC HPSUMMARY provides functionality similar to that of the SUMMARY procedure in Base SAS. Its syntax, options, and underlying concepts are also similar. Because of this similarity, documentation for the SUMMARY procedure can be useful in understanding PROC HPSUMMARY. For more information about the SUMMARY procedure, see the *Base SAS Procedures Guide*.

PROC HPSUMMARY Features

PROC HPSUMMARY provides data summarization tools to compute descriptive statistics for variables across all observations and within groups of observations. For example, PROC HPSUMMARY does the following:

- calculates descriptive statistics based on moments
- calculates and estimates quantiles, which includes the median
- calculates confidence limits for the mean
- identifies extreme values
- performs a *t* test

PROC HPSUMMARY does not display output. You can use the OUTPUT statement to store the statistics in a SAS data set.

PROC HPSUMMARY provides a vehicle for the parallel execution of summarization in a distributed computing environment. The following list summarizes the basic features of PROC HPSUMMARY:

- provides the ability to execute summarization in parallel
- enables you to control the level of parallelism per execution node and the number of nodes to engage
- is highly multithreaded
- manages data migration to the location of execution and movement back to the client machine as needed

Because the HPSUMMARY procedure is a high-performance analytical procedure, it also does the following:

- enables you to run in distributed mode on a cluster of machines that distribute the data and the computations
- enables you to run in single-machine mode on the server where SAS is installed
- exploits all the available cores and concurrent threads, regardless of execution mode

For more information, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

Single-Machine and Distributed Execution Modes

The HPSUMMARY procedure enables you to perform analyses either on a single computer (single-machine mode) or on multiple computers that are connected in a grid configuration (distributed mode). For more information about these execution modes, see the section "Processing Modes" on page 10 in Chapter 3, "Shared Concepts and Topics."

In single-machine mode, you can take advantage of multiple processors and cores in a single machine, and you can control the number of parallel threads.

In distributed mode, you can take advantage of the collective processing resources of multiple machines. You can control both the number of parallel threads per execution node and the number of nodes to engage. One or more copies of the summarization code are executed in parallel on each node. You can read data in parallel from and write data in parallel to a supported database management system (DBMS) on each node in the grid, thus greatly reducing processing time for large volumes of data. The distributed mode of execution has two variations:

- In the client-data (local-data) model of distributed execution, the input data are not stored on the grid computing appliance but are distributed to it from the client during execution of the HPSUMMARY procedure.
- In the alongside-the-database model of distributed execution, the data source is the database on the appliance. The data are stored in the distributed database, and the summarization code that runs on each node can read and write the data in parallel during execution of the procedure. Instead of being moved across the network and possibly back to the client machine, data are passed locally between the processes on each node of the appliance. In general, especially with large data sets, the best PROC HPSUMMARY performance can be achieved if execution is alongside the database.

PROC HPSUMMARY Contrasted with Other Procedures

By default, PROC SUMMARY generates all CLASS variable combination types and requires the NWAY option to generate only the *n*-way. By default, PROC HPSUMMARY generates only the *n*-way, and requires the ALLTYPES option to generate all of the types.

Getting Started: HPSUMMARY Procedure

This example illustrates a simple use of the HPSUMMARY procedure to summarize using the grid in distributed mode. The following DATA step creates a data set that consists of test scores:

```
data gridlib.grades;
   input Name $ 1-8 Gender $ 11 Status $13 Year $ 15-16
         Section $ 18 Score 20-21 FinalGrade 23-24;
   datalines;
Abbott    F 2 97 A 90 87
```

```
Branford  M 1 98 A 92 97
Crandell  M 2 98 B 81 71
Dennison  M 1 97 A 85 72
Edgar     F 1 98 B 89 80
Faust     M 1 97 B 78 73
Greeley   F 2 97 A 82 91
Hart      F 1 98 B 84 80
Isley     M 2 97 A 88 86
Jasper    M 1 97 B 91 93
;
run;
```

The following statements read this data set and analyze the data for the two-way combination of CLASS variables and across all observations. To run these statements successfully, you need to set the macro variables GRIDHOST and GRIDINSTALLLOC to resolve to appropriate values, or you can replace the references to the macro variables in the example with the appropriate values.

```
proc hpsummary data=gridlib.grades;
   performance host="&GRIDHOST" install="&GRIDINSTALLLOC";
   var Score;
   class Status Year;
   types () status*year;
   output out=gridlib.result;
run;
proc print data=gridlib.result;
run;
```

Figure 10.1 displays the tables produced by the HPSUMMARY procedure. The "Performance Information" table shows that PROC HPSUMMARY ran in distributed mode and the "Data Access Information" table shows that the date were accessed in parallel symmetric mode.

Figure 10.1 HPSUMMARY Output

The HPSUMMARY Procedure

Performance Information	
Host Node	greenarrow.unx.sas.com
Execution Mode	Distributed
Number of Compute Nodes	16
Number of Threads per Node	24

Data Access Information			
Data	**Engine**	**Role**	**Path**
GRIDLIB.grades	GREENPLM	Input	Parallel, Symmetric
GRIDLIB.result	GREENPLM	Output	Parallel, Symmetric

Figure 10.1 *continued*

Obs	status	year	_type_	_freq_	_stat_	score
1	1	98	3	3	MIN	84.0000
2			0	10	STD	4.7140
3			0	10	MAX	92.0000
4			0	10	MIN	78.0000
5	1	98	3	3	MAX	92.0000
6	2	98	3	1	MIN	81.0000
7	2	97	3	3	N	3.0000
8	1	97	3	3	STD	6.5064
9	1	98	3	3	N	3.0000
10	2	97	3	3	MAX	90.0000
11			0	10	N	10.0000
12	2	98	3	1	MEAN	81.0000
13	2	98	3	1	MAX	81.0000
14	2	97	3	3	MIN	82.0000
15	2	98	3	1	STD	.
16	1	97	3	3	N	3.0000
17	2	97	3	3	STD	4.1633
18	1	97	3	3	MEAN	84.6667
19	2	97	3	3	MEAN	86.6667
20	1	97	3	3	MAX	91.0000
21	1	97	3	3	MIN	78.0000
22			0	10	MEAN	86.0000
23	1	98	3	3	STD	4.0415
24	1	98	3	3	MEAN	88.3333
25	2	98	3	1	N	1.0000

Syntax: HPSUMMARY Procedure

The following statements are available in the HPSUMMARY procedure:

PROC HPSUMMARY < *options* > < *statistic-keywords* > **;**
CLASS *variables* < / *options* > **;**
FREQ *variable* **;**
OUTPUT < **OUT=***SAS-data-set* > < *output-statistic-specifications* > < / **AUTONAME** > **;**
PERFORMANCE *performance-options* **;**
TYPES *requests* **;**
VAR *variables* < / **WEIGHT=***weight-variable* > **;**
WAYS *list* **;**
WEIGHT *variable* **;**

You can also use the ATTRIB, FORMAT, LABEL, and WHERE statements and any global statements. For more information, see *SAS Statements: Reference*.

PROC HPSUMMARY Statement

PROC HPSUMMARY < *options* > < *statistic-keywords* > **;**

The PROC HPSUMMARY statement invokes the procedure. The HPSUMMARY procedure computes descriptive statistics for variables across all observations or within groups of observations.

Table 10.1 summarizes the available *options* in the PROC HPSUMMARY statement by function. The *options* are then described fully in alphabetical order in the section "Optional Arguments" on page 136. For information about the *statistic-keywords*, see the section "Statistic Keywords" on page 139.

Table 10.1 PROC HPSUMMARY Statement Options

Option	Description
Basic Options	
DATA=	Specifies the input data set
PCTLDEF=	Specifies the mathematical definition used to compute quantiles
Option Related to Classification Level	
MISSING	Uses missing values as valid values to create combinations of classification variables
Options Related to the Output Data Set	
ALLTYPES	Computes statistics for all combinations of classification variables (not just the *n*-way)
CHARTYPE	Specifies that the _TYPE_ variable contain character values
Options Related to Statistical Analysis	
ALPHA=	Specifies the confidence level for the confidence limits
EXCLNPWGT	Excludes observations with nonpositive weights from the analysis
QMARKERS=	Specifies the sample size to use for the P2 quantile estimation method
QMETHOD=	Specifies the quantile estimation method
QNTLDEF=	Specifies the mathematical definition used to compute quantiles
statistic-keywords	Selects the statistics
VARDEF=	Specifies the variance divisor

Optional Arguments

You can specify the following *options* in the PROC HPSUMMARY statement:

ALLTYPES

ALLWAYS

requests that PROC HPSUMMARY compute descriptive statistics for all combinations of classification variables. By default, PROC HPSUMMARY generates only the *n*-way. For more information, see the section "How PROC HPSUMMARY Groups Data" on page 147.

ALPHA=*value*

specifies the confidence level to compute the confidence limits for the mean. The percentage for the confidence limits is 100(1–*value*). For example, ALPHA=0.05 results in a 95% confidence limit. You

can specify any *value* between 0 and 1. The default is 0.05. To compute confidence limits, specify the *statistic-keyword* CLM, LCLM, or UCLM. See the section "Confidence Limits" on page 149.

CHARTYPE
specifies that the _TYPE_ variable in the output data set is a character representation of the binary value of _TYPE_. The length of the variable equals the number of classification variables. When you specify more than 32 classification variables, _TYPE_ automatically becomes a character variable. See the section "Output Data Set" on page 152.

DATA=*SAS-data-set*
names the SAS data set to be used as the input data set. The default is the most recently created data set.

EXCLNPWGT
EXCLNPWGTS
excludes observations with nonpositive weight values (0 or negative) from the analysis. By default, PROC HPSUMMARY treats observations with negative weights like observations with zero weights and counts them in the total number of observations. See the WEIGHT= option and the section "WEIGHT Statement" on page 146.

MISSING
considers missing values as valid values to create the combinations of classification variables. Special missing values that represent numeric values—the letters A through Z and the underscore (_) character—are each considered as a separate value. If you omit MISSING, then PROC HPSUMMARY excludes the observations with a missing classification variable value from the analysis. See *SAS Language Reference: Concepts* for a discussion of missing values that have special meanings.

PCTLDEF=1 | 2 | 3 | 4 | 5
is an alias for the QNTLDEF= option.

QMARKERS=*number*
specifies the default number of markers to use for the P^2 quantile estimation method. The number of markers controls the size of fixed memory space.

The value of *number* must be an odd integer greater than 3. The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quantiles (P25 and P50), *number* is 25. For the quantiles P1, P5, P10, P75 P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC HPSUMMARY uses the largest value of *number*.

You can improve the accuracy of the estimate by increasing the number of markers above the default settings; you can conserve memory and computing time by reducing the number of markers. See the section "Quantiles" on page 150.

QMETHOD=OS | P2
specifies the method that PROC HPSUMMARY uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the QMARKERS= value and QNTLDEF=5, then both methods produce the same results. The QMETHOD= option can take either of the following values:

OS specifies that PROC HPSUMMARY use order statistics.

NOTE: This technique can be very memory-intensive.

P2 specifies that PROC HPSUMMARY use the P^2 method to approximate the quantile. When QMETHOD=P2, PROC HPSUMMARY does not compute MODE or weighted quantiles. In addition, reliable estimations of some quantiles (P1, P5, P95, P99) might not be possible for some data sets.

The default is OS. See the section "Quantiles" on page 150 for more information.

QNTLDEF=1 | 2 | 3 | 4 | 5

PCTLDEF=1 | 2 | 3 | 4 | 5

specifies the mathematical definition that PROC HPSUMMARY uses to calculate quantiles when QMETHOD=OS. The default is 5. To use QMETHOD=P2, you must use QNTLDEF=5. See the section "Quantile and Related Statistics" on page 157.

VARDEF=*divisor*

specifies the divisor to use in the calculation of the variance and standard deviation. Table 10.2 shows the possible values for *divisor* and their associated formulas.

Table 10.2 Values for VARDEF= Option

divisor	**Description**	**Formula for Divisor**
DF	Degrees of freedom	$n - 1$
N	Number of observations	n
WDF	Sum of weights minus one	$\left(\sum_i w_i\right) - 1$
WEIGHT | WGT	Sum of weights	$\sum_i w_i$

The procedure computes the variance as CSS divided by *divisor*, where the corrected sum of squares CSS is defined by the following formula:

$$\text{CSS} = \sum (x_i - \bar{x})^2$$

When you weight the analysis variables, the formula for CSS is

$$\text{CSS} = \sum w_i \, (x_i - \bar{x}_w)^2$$

where $\bar{x}_w$ is the weighted mean.

The default is DF. To compute the standard error of the mean, confidence limits for the mean, or the Student's *t*-test, you must use this default value.

When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the *i*th observation is $\text{var}(x_i) = \sigma^2/w_i$ and w_i is the weight for the *i*th observation. This method yields an estimate of the variance of an observation with unit weight. When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of $\sigma^2/\bar{w}$, where $\bar{w}$ is the average weight. This method yields an asymptotic estimate of the variance of an observation with average weight. See the section "Keywords and Formulas" on page 153.

Statistic Keywords

Optional *statistic-keywords* specify which statistics to compute and the order to display them in the output. Table 10.3 lists the keywords that are available in the PROC HPSUMMARY statement. The definitions of the keywords and the formulas for the associated statistics are listed in the section "Keywords and Formulas" on page 153.

Table 10.3 Statistic Keywords in the PROC HPSUMMARY Statement

Descriptive Statistic Keywords:	
CLM	NMISS
CSS	RANGE
CV	SKEWNESS | SKEW
KURTOSIS | KURT	STDDEV | STD
LCLM	STDERR
MAX	SUM
MEAN	SUMWGT
MIN	UCLM
MODE	USS
N	VAR
Quantile Statistic Keywords:	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
P20	P30
P40	P60
P70	P80
Q1 | P25	QRANGE
Hypothesis Testing Keywords:	
PROBT | PRT	T

The default values are N, MEAN, STD, MIN, and MAX. To compute standard error, confidence limits for the mean, and the Student's *t*-test, you must use the default value of the VARDEF= option, which is DF. To compute skewness or kurtosis, you must use VARDEF=N or VARDEF=DF. Use CLM or both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM to compute a one-sided confidence limit. The definitions of the keywords and the formulas for the associated statistics are listed in the section "Keywords and Formulas" on page 153.

CLASS Statement

CLASS *variables* < / *options* > **;**

The CLASS statement names the classification variables to be used as explanatory variables in the analysis. These variables enter the analysis not through their values, but through levels to which the unique values are mapped. For more information, see the section "Levelization of Classification Variables" (Chapter 4, *SAS/STAT User's Guide: High-Performance Procedures*).

Levels of classification variables are ordered by their external formatted values, except for numeric variables with no explicit format, which are ordered by their unformatted (internal) values.

Required Argument

variables

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are referred to as *classification variables*. Classification variables are numeric or character. Classification variables can have continuous values, but they typically have a few discrete values that define levels of the variable. You do not have to sort the data by classification variables.

Use the TYPES statement or the WAYS statement to control which classification variables PROC HPSUMMARY uses to group the data. See the section "How PROC HPSUMMARY Groups Data" on page 147.

To reduce the number of classification variable levels, use a FORMAT statement to combine variable values. When a format combines several internal values into one formatted value, PROC HPSUMMARY outputs the lowest internal value.

Optional Arguments

GROUPINTERNAL

specifies that formats are not to be applied to the classification variables when PROC HPSUMMARY groups the values to create combinations of classification variables. This option saves computer resources when the numeric classification variables contain discrete values. See the section "Computational Resources" on page 148.

MISSING

considers missing values as valid values for the classification variable levels. Special missing values that represent numeric values—the letters A through Z and the underscore (_) character—are each considered as a separate value. If you omit the MISSING option, then PROC HPSUMMARY excludes the observations with a missing classification variable value from the analysis.

By default, if an observation contains a missing value for any classification variable, then PROC HPSUMMARY excludes that observation from the analysis. If you specify the MISSING option in the PROC HPSUMMARY statement, then the procedure considers missing values as valid levels for the combination of classification variables.

Specifying the MISSING option in the CLASS statement enables you to control the acceptance of missing values for individual classification variables.

See *SAS Language Reference: Concepts* for a discussion of missing values that have special meaning.

FREQ Statement

FREQ *variable* **;**

The FREQ statement specifies a numeric variable that contains the frequency of each observation.

Required Argument

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents *n* observations, where *n* is the value of *variable*. If *n* is not an integer, then SAS truncates it. If *n* is less than 1 or is missing, then the procedure does not use that observation to calculate statistics.

The sum of the frequency variable represents the total number of observations.

OUTPUT Statement

OUTPUT < **OUT=***SAS-data-set* > < *output-statistic-specifications* > < / **AUTONAME** > **;**

The OUTPUT statement writes statistics to a new SAS data set. You can use multiple OUTPUT statements to create several OUT= data sets.

Optional Arguments

OUT=*SAS-data-set*

names the new output data set. If *SAS-data-set* does not exist, then PROC HPSUMMARY creates it. If you omit the OUT= option, then the data set is named DATA*n*, where *n* is the smallest integer that makes the name unique.

output-statistic-specifications

specifies the statistics to store in the OUT= data set and names one or more variables that contain the statistics. The form of the *output-statistic-specification* is

statistic-keyword < **(***variable-list***)** >**=**< *names* >

where

statistic-keyword

specifies which statistic to store in the output data set. Table 10.4 lists the *statistic-keywords* that are available in the OUTPUT statement.

Table 10.4 Statistics Keywords in the OUTPUT Statement

Descriptive Statistic Keywords:	
CSS	RANGE
CV	SKEWNESS | SKEW
KURTOSIS | KURT	STDDEV | STD
LCLM	STDERR

Table 10.4 (continued)

MAX	SUM
MEAN	SUMWGT
MIN	UCLM
MODE	USS
N	VAR
NMISS	
Quantile Statistic Keywords:	
MEDIAN | P50	Q3 | P75
P1	P90
P5	P95
P10	P99
P20	P30
P40	P60
P70	P80
Q1 | P25	QRANGE
Hypothesis Testing Keywords:	
PROBT | PRT	T

By default the statistics in the output data set automatically inherit the analysis variable's format, informat, and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this format might be invalid for these statistics (for example, dollar or datetime formats). If you omit a *variable-list* and *names*, then PROC HPSUMMARY allows the *statistic-keyword* only once in a single OUTPUT statement, unless you also use the AUTONAME option.

The definitions of the keywords and the formulas for the associated statistics are listed in the section "Keywords and Formulas" on page 153.

variable-list

specifies the names of one or more numeric analysis variables whose statistics you want to store in the output data set. By default, statistics are stored for all numeric analysis variables.

names

specifies one or more names for the variables in output data set to contain the analysis variable statistics. The first name contains the statistic for the first analysis variable; the second name contains the statistic for the second analysis variable; and so on. The default value is the analysis variable name. If you specify the AUTONAME option, then the default is the combination of the analysis variable name and the *statistic-keyword*. If you use the CLASS statement and an OUTPUT statement without an *output-statistic-specification*, then the output data set contains five observations for each combination of classification variables: the value of N, MIN, MAX, MEAN, and STD. If you use the WEIGHT statement or the WEIGHT option in the VAR statement, then the output data set also contains an observation with the sum of weights (SUMWGT) for each combination of classification variables.

If you specify *variable-list*, then PROC HPSUMMARY uses the order in which you specify the analysis variables to store the statistics in the output data set variables. You can use the

AUTONAME option to request that PROC HPSUMMARY generate unique names for multiple variables and statistics.

AUTONAME
requests that PROC HPSUMMARY create a unique variable name for an output statistic when you do not assign the variable name in the OUTPUT statement. This action is accomplished by appending the *statistic-keyword* to the input variable name. For example, the following statement produces the x_Min variable in the output data set:

```
output min(x)=/autoname;
```

AUTONAME activates the SAS internal mechanism that automatically resolves conflicts in the variable names in the output data set so that duplicate variables do not generate errors. As a result, the following statement produces two variables, x_Min and x_Min2, in the output data set:

```
output min(x)= min(x)=/autoname;
```

PERFORMANCE Statement

PERFORMANCE *performance-options* **;**

The PERFORMANCE statement defines performance parameters for multithreaded and distributed computing, passes variables that describe the distributed computing environment, and requests detailed results about the performance characteristics of the HPSUMMARY procedure.

You can also use the PERFORMANCE statement to control whether PROC HPSUMMARY executes in single-machine or distributed mode.

It is important to remember the distinction between the NODES= and NTHREADS= options. The NODES= option specifies the number of separate grid nodes that participate in the PROC HPSUMMARY execution, and the NTHREADS= option influences how many threads are used by the HPSUMMARY procedure instance that runs on each node. If the data are located on the grid, then all nodes must be engaged; therefore, the NODES= option might be overridden. Specifying NODES=0 causes PROC HPSUMMARY to execute in single-machine mode only. Setting the NTHREADS= option to a value that is greater than the CPU count on each grid node is not likely to improve overall throughput.

The PERFORMANCE statement is documented further in the section "PERFORMANCE Statement" on page 36 of Chapter 3, "Shared Concepts and Topics."

TYPES Statement

TYPES *requests* **;**

The TYPES statement identifies which of the possible combinations of classification variables to generate. The TYPES statement requires the specification of a CLASS statement.

Required Argument

requests

specifies which of the 2^k combinations of classification variables PROC HPSUMMARY uses to create the types, where k is the number of classification variables. A *request* includes one classification variable name, several classification variable names separated by asterisks, or ().

To request classification variable combinations quickly, use a grouping syntax by placing parentheses around several variables and joining other variables or variable combinations. The examples in Table 10.5 illustrate grouping syntax:

Table 10.5 Examples of Grouping Syntax

Request	Equivalent To
`types A*(B C);`	`types A*B A*C;`
`types (A B)*(C D);`	`types A*C A*D B*C B*D;`
`types (A B C)*D;`	`types A*D B*D C*D;`

You can use parentheses () to request the overall total (_TYPE_=0). If you do not need all types in the output data set, then use the TYPES statement to specify particular subtypes rather than applying a WHERE clause to the data set. Doing so saves time and computer memory.

Order of Analyses in the Output

The SUMMARY procedure writes analyses to the output in order of increasing values of the _TYPE_ variable. When PROC HPSUMMARY executes on the grid, the order of observations within the output is not deterministic because the output is returned in parallel. You can sort the output as follows:

- If output is directed back to the client, then to achieve an output order that is similar to the output of PROC SUMMARY, you need to subsequently sort the data by _TYPE_ and the classification variables.
- If output is directed back to the grid (so that the results are distributed), then there is no order within the output. To retrieve the observations in order, you can execute an SQL query, specifying that the selecting rows be returned in order by _TYPE_ and the classification variables.

The _TYPE_ variable is calculated even if no output data set is requested. For more information about the _TYPE_ variable, see the section "Output Data Set" on page 152.

VAR Statement

VAR *variables* < / **WEIGHT=***weight-variable* > **;**

The VAR statement identifies the analysis variables and their order in the output. If you omit the VAR statement, then PROC HPSUMMARY analyzes all numeric variables that are not listed in the other statements. When all variables are character variables, PROC SUMMARY produces a simple count of observations. You can use multiple VAR statements.

Required Argument

variables
identifies one or more analysis variables and specifies their order in the results.

Optional Argument

WEIGHT=*weight-variable*
specifies a numeric variable whose values weight the values of the *variables*. The *weight-variable* does not have to be an integer. Table 10.6 describes how PROC HPSUMMARY treats various values of the *weight-variable*.

Table 10.6 Responses to Values of *weight-variable*

Value	PROC HPSUMMARY Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use the EXCLNPWGT option in the PROC HPSUMMARY statement.

The *weight-variable* does not change how the procedure determines the range, extreme values, or number of missing values.

To compute weighted quantiles, use QMETHOD=OS in the PROC HPSUMMARY statement. Skewness and kurtosis are not available with the WEIGHT= option.

When you use the WEIGHT= option, consider which value of the VARDEF= option is appropriate. Use the WEIGHT= option in multiple VAR statements to specify different weights for the analysis variables.

WAYS Statement

WAYS *list* **;**

The WAYS statement specifies the number of ways to make unique combinations of classification variables. You can use the TYPES statement to specify additional combinations of classification variables.

Required Argument

list

specifies one or more integers that define the number of classification variables to combine to form all the unique combinations of classification variables. For example, you can specify 2 for all possible pairs and 3 for all possible triples. The *list* can be specified in the following ways:

- *m*
- *m1 m2 ... mn*
- *m1*,*m2*,...,*mn*
- *m* TO *n* <BY *increment*>
- *m1*,*m2*, TO *m3* <BY *increment*>,*m4*

The range of *list* is from 0 to the maximum number of classification variables.

The following statements are an example of creating two-way types for the classification variables A, B, and C:

```
class A B C ;
ways 2;
```

The WAYS statement in this example is equivalent to specifying **A*B**, **A*C**, and **B*C** in the TYPES statement.

WEIGHT Statement

WEIGHT *weight-variable* **;**

The WEIGHT statement specifies weights for observations in the statistical calculations.

Required Argument

weight-variable

specifies a numeric *weight-variable* whose values weight the values of the analysis variables. The values of *weight-variable* do not have to be integers. Table 10.7 describes how PROC HPSUMMARY treats various values of *weight-variable*.

Table 10.7 Responses to Values of *weight-variable*

Value	PROC HPSUMMARY Response
0	Counts the observation in the total number of observations
Less than 0	Converts the value to zero and counts the observation in the total number of observations
Missing	Excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use the EXCLNPWGT option in the PROC HPSUMMARY statement.

CAUTION: Single extreme weight values can cause inaccurate results. When one (and only one) weight value is many orders of magnitude larger than the other weight values (for example, 49 weight values of 1 and one weight value of 1×10^{14}), certain statistics might not be within acceptable accuracy limits. The affected statistics are based on the second moment (such as standard deviation, corrected sum of squares, variance, and standard error of the mean). Under certain circumstances, no warning is written to the SAS log.

To compute weighted quantiles, you must use QMETHOD=OS in the PROC statement. Skewness and kurtosis are not available with the WEIGHT statement.

PROC HPSUMMARY does not compute MODE when a weight variable is active. Instead, you can try using the UNIVARIATE procedure when MODE needs to be computed and a weight variable is active.

If you use the WEIGHT= option in a VAR statement to specify a weight variable, then PROC HPSUMMARY uses this variable instead to weight those VAR statement variables.

When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the section "Keywords and Formulas" on page 153 for more information.

Details: HPSUMMARY Procedure

How PROC HPSUMMARY Groups Data

Groups of observations are defined by specifying certain variables as classification variables in the CLASS statement. Unique values of the n CLASS variables are used to partition the input data, and the resulting summarized data (one observation per group) is called the "n-way."

PROC HPSUMMARY can also combine the partitioned groups into larger groups by removing one or more CLASS variables from consideration when grouping. There are 2^n different groupings that can be generated from n CLASS variables. Each of these groupings is a "type," which appears in the output data set as a variable named _TYPE_. Type 0 includes no CLASS variables and summarizes the entire input data set, Type 1 includes only the last CLASS variable specified, and so on to Type $2^n - 1$, which is the n-way.

By default, PROC HPSUMMARY generates only the n-way. The option ALLTYPES (or ALLWAYS) in the PROC HPSUMMARY statement generates all 2^n types. You can also use either of the following statements to choose which types appear in the output data set:

- The WAYS statement specifies how many CLASS variables appear in each output type. For example, WAYS 1 produces types for each CLASS variable individually, WAYS 2 generates all $\binom{n}{2}$ possible pairs, and so on.
- The TYPES statement explicitly specifies the desired types by CLASS variable name, such as TYPES A A*B C (), where A*B might specify Type 6 and "()" specifies Type 0.

The TYPES statement controls which of the available classification variables PROC HPSUMMARY uses to subgroup the data. The unique combinations of these active classification variable values that occur

together in any single observation of the input data set determine the data subgroups. Each subgroup that PROC HPSUMMARY generates for a given type is called a level of that type. For all types, the inactive classification variables can still affect the total observation count of the rejection of observations with missing values. When you use a WAYS statement, PROC HPSUMMARY generates types that correspond to every possible unique combination of n classification variables chosen from the complete set of classification variables. For example

```
proc hpsummary;
   class a b c d e;
   ways 2 3;
   output out=results;
run;
```

is equivalent to

```
proc hpsummary;
   class a b c d e;
   types a*b a*c a*d a*e b*c b*d b*e c*d c*e d*e
         a*b*c a*b*d a*b*e a*c*d a*c*e a*d*e
         b*c*d b*c*e c*d*e;
   output out=results;
run;
```

If you omit the TYPES statement and the WAYS statement, then PROC HPSUMMARY uses all classification variables to subgroup the data (the NWAY type) for the output data set.

Computational Resources

The total of unique classification values that PROC HPSUMMARY allows depends on the amount of computer memory that is available. PROC HPSUMMARY uses the same memory allocation scheme across all operating environments. When classification variables are involved, PROC HPSUMMARY must keep a copy of each unique value of each classification variable in memory. You can estimate the memory requirements to group the classification variable by calculating

$$Nc_1(Lc_1 + K) + Nc_2(Lc_2 + K) + \ldots + Nc_n(Lc_n + K)$$

where Nc_i is the number of unique values for the classification variable, Lc_i is the combined unformatted and formatted length of c_i, and K is some constant on the order of 32 bytes (64 for 64-bit architectures). When you use the GROUPINTERNAL option in the CLASS statement, Lc_i is simply the unformatted length of c_i.

The GROUPINTERNAL option can improve computer performance because the grouping process is based on the internal values of the classification variables. If a numeric classification variable is not assigned a format and you do not specify GROUPINTERNAL, then PROC HPSUMMARY uses the default format, BEST12., to format numeric values as character strings. Then PROC HPSUMMARY groups these numeric variables by their character values, which takes additional time and computer memory.

Each unique combination of classification variables $c_{1_i}\ c_{2_j}$ for a given type forms a level in that type. See the section "TYPES Statement" on page 143. You can estimate the maximum potential space requirements for all levels of a given type, when all combinations actually exist in the data (a complete type), by calculating

$$W * Nc_1 * Nc_2 * \ldots * Nc_n$$

where W is a constant based on the number of variables analyzed and the number of statistics calculated (unless you request QMETHOD=OS to compute the quantiles) and $Nc_1 \ldots Nc_n$ are the number of unique levels for the active classification variables of the given type.

Clearly, the memory requirements of the levels overwhelm the levels of the classification variables. For information about how to adjust your computation resource parameters, see the SAS documentation for your operating environment.

Another way to enhance performance is by carefully applying the TYPES or WAYS statement, limiting the computations to only those combinations of classification variables that you are interested in.

Statistical Computations

Computation of Moment Statistics

PROC HPSUMMARY uses single-pass algorithms to compute the moment statistics (such as mean, variance, skewness, and kurtosis). See the section "Keywords and Formulas" on page 153 for the statistical formulas.

The computational details for confidence limits, hypothesis test statistics, and quantile statistics follow.

Confidence Limits

With the *statistic-keywords* CLM, LCLM, and UCLM, you can compute confidence limits for the mean. A confidence limit is a range (constructed around the value of a sample statistic) that contains the corresponding true population value with given probability (ALPHA=) in repeated sampling. A two-sided $100(1-\alpha)\%$ confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1-\alpha/2;n-1)} \frac{s}{\sqrt{n}}$$

where $s = \sqrt{\frac{1}{n-1}\sum(x_i - \bar{x})^2}$ and $t_{(1-\alpha/2;n-1)}$ is the $(1-\alpha/2)$ critical value of the Student's t statistic with $n-1$ degrees of freedom.

A one-sided $100(1-\alpha)\%$ confidence interval is computed as

$$\bar{x} + t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} \qquad \text{(upper)}$$

$$\bar{x} - t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} \qquad \text{(lower)}$$

A two-sided $100(1-\alpha)\%$ confidence interval for the standard deviation has lower and upper limits

$$s\sqrt{\frac{n-1}{\chi^2_{(1-\alpha/2;n-1)}}}, \quad s\sqrt{\frac{n-1}{\chi^2_{(\alpha/2;n-1)}}}$$

where $\chi^2_{(1-\alpha/2;n-1)}$ and $\chi^2_{(\alpha/2;n-1)}$ are the $(1-\alpha/2)$ and $\alpha/2$ critical values of the chi-square statistic with $n-1$ degrees of freedom. A one-sided $100(1-\alpha)\%$ confidence interval is computed by replacing $\alpha/2$ with α.

A $100(1-\alpha)\%$ confidence interval for the variance has upper and lower limits that are equal to the squares of the corresponding upper and lower limits for the standard deviation.

If you use the WEIGHT statement or the WEIGHT= option in a VAR statement and the default value of the VARDEF= option (which is DF), the $100(1-\alpha)\%$ confidence interval for the weighted mean has upper and lower limits

$$\bar{y}_w \pm t_{(1-\alpha/2)} \frac{s_w}{\sqrt{\sum_{i=1}^{n} w_i}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, w_i is the weight for the ith observation, and $t_{(1-\alpha/2)}$ is the $(1-\alpha/2)$ critical value for the Student's t distribution with $n-1$ degrees of freedom.

Student's *t* Test

PROC HPSUMMARY calculates the t statistic as

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

where $\bar{x}$ is the sample mean, n is the number of nonmissing values for a variable, and s is the sample standard deviation. Under the null hypothesis, the population mean equals μ_0. When the data values are approximately normally distributed, the probability under the null hypothesis of a t statistic as extreme as, or more extreme than, the observed value (the p-value) is obtained from the t distribution with $n-1$ degrees of freedom. For large n, the t statistic is asymptotically equivalent to a z test.

When you use the WEIGHT statement or the WEIGHT= option in a VAR statement and the default value of the VARDEF= option (which is DF), the Student's t statistic is calculated as

$$t_w = \frac{\bar{y}_w - \mu_0}{s_w \Big/ \sqrt{\sum_{i=1}^{n} w_i}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, and w_i is the weight for the ith observation. The t_w statistic is treated as having a Student's t distribution with $n-1$ degrees of freedom. If you specify the EXCLNPWGT option in the PROC HPSUMMARY statement, then n is the number of nonmissing observations when the value of the WEIGHT variable is positive. By default, n is the number of nonmissing observations for the WEIGHT variable.

Quantiles

The options QMETHOD=, QNTLDEF=, and QMARKERS= determine how PROC HPSUMMARY calculates quantiles. The QNTLDEF= option deals with the mathematical definition of a quantile. See the section "Quantile and Related Statistics" on page 157. The QMETHOD= option specifies how PROC HPSUMMARY handles the input data: When QMETHOD=OS, PROC HPSUMMARY reads all data into memory and sorts it by unique value. When QMETHOD=P2, PROC HPSUMMARY accumulates all data into a fixed sample size that is used to approximate the quantile.

If data set A has 100 unique values for a numeric variable X and data set B has 1,000 unique values for numeric variable X, then QMETHOD=OS for data set B requires 10 times as much memory as it does for data set A. If QMETHOD=P2, then both data sets A and B require the same memory space to generate quantiles.

The QMETHOD=P2 technique is based on the piecewise-parabolic (P^2) algorithm invented by Jain and Chlamtac (1985). P^2 is a one-pass algorithm to determine quantiles for a large data set. It requires a fixed amount of memory for each variable for each level within the type. However, using simulation studies, reliable estimations of some quantiles (P1, P5, P95, P99) cannot be possible for some data sets such as data sets with heavily tailed or skewed distributions.

If the number of observations is less than the QMARKERS= value, then QMETHOD=P2 produces the same results as QMETHOD=OS when QNTLDEF=5. To compute weighted quantiles, you must use QMETHOD=OS.

Results

Missing Values

PROC HPSUMMARY excludes missing values for the analysis variables before calculating statistics. Each analysis variable is treated individually; a missing value for an observation in one variable does not affect the calculations for other variables. PROC HPSUMMARY handles missing values as follows:

- If a classification variable has a missing value for an observation, then PROC HPSUMMARY excludes that observation from the analysis unless you use the MISSING option in the PROC statement or CLASS statement.
- If a FREQ variable value is missing or nonpositive, then PROC HPSUMMARY excludes the observation from the analysis.
- If a WEIGHT variable value is missing, then PROC HPSUMMARY excludes the observation from the analysis.

PROC HPSUMMARY tabulates the number of the missing values. Before the number of missing values are tabulated, PROC HPSUMMARY excludes observations with frequencies that are nonpositive when you use the FREQ statement and observations with weights that are missing or nonpositive (when you use the EXCLNPWGT option) when you use the WEIGHT statement. To report this information in the procedure output use the NMISS *statistic-keyword* in the PROC HPSUMMARY statement.

The N Obs Statistic

By default when you use a CLASS statement, PROC HPSUMMARY displays an additional statistic called N Obs. This statistic reports the total number of observations or the sum of the observations of the FREQ variable that PROC HPSUMMARY processes for each class level. PROC HPSUMMARY might omit observations from this total because of missing values in one or more classification variables. Because of this action and the exclusion of observations when the *weight-variable* (specified in the WEIGHT statement or in the WEIGHT= option in the VAR statement) contains missing values, there is not always a direct relationship between N Obs, N, and NMISS.

In the output data set, the value of N Obs is stored in the _FREQ_ variable.

Output Data Set

PROC HPSUMMARY creates one output data set. The procedure does not print the output data set. Use the PRINT procedure, the REPORT procedure, or another SAS reporting tool to display the output data set.

NOTE: By default, the statistics in the output data set automatically inherit the analysis variable's format and label. However, statistics computed for N, NMISS, SUMWGT, USS, CSS, VAR, CV, T, PROBT, PRT, SKEWNESS, and KURTOSIS do not inherit the analysis variable's format because this format can be invalid for these statistics.

The output data set can contain these variables:

- the variables specified in the CLASS statement.
- the variable _TYPE_ that contains information about the classification variables. By default _TYPE_ is a numeric variable. If you specify CHARTYPE in the PROC statement, then _TYPE_ is a character variable. When you use more than 32 classification variables, _TYPE_ is automatically a character variable.
- the variable _FREQ_ that contains the number of observations that a given output level represents.
- the variables requested in the OUTPUT statement that contain the output statistics and extreme values.
- the variable _STAT_ that contains the names of the default statistics if you omit statistic keywords.

The value of _TYPE_ indicates which combination of the classification variables PROC HPSUMMARY uses to compute the statistics. The character value of _TYPE_ is a series of zeros and ones, where each value of one indicates an active classification variable in the type. For example, with three classification variables, PROC HPSUMMARY represents type 1 as 001, type 5 as 101, and so on.

Usually, the output data set contains one observation per level per type. However, if you omit *statistic-keywords* in the OUTPUT statement, then the output data set contains five observations per level (six if you specify a WEIGHT variable). Therefore, the total number of observations in the output data set is equal to the sum of the levels for all the types that you request multiplied by 1, 5, or 6, whichever is applicable.

If you omit the CLASS statement (_TYPE_= 0), then there is always exactly one level of output per output data set. If you use a CLASS statement, then the number of levels for each type that you request has an upper bound equal to the number of observations in the input data set. By default, PROC HPSUMMARY generates all possible types. In this case the total number of levels for each output data set has an upper bound equal to

$$m \cdot (2^k - 1) \cdot n + 1$$

where k is the number of classification variables, n is the number of observations in the input data set, and m is 1, 5, or 6.

PROC HPSUMMARY determines the actual number of levels for a given type from the number of unique combinations of each active classification variable. A single level consists of all input observations whose formatted class values match.

Table 10.8 shows the values of _TYPE_ and the number of observations in the data set when you specify one, two, and three classification variables.

Table 10.8 The Effect of Classification Variables on the OUTPUT Data Set

CLASS Variables							
C	B	A	_WAY_	_TYPE_	Subgroup defined by	Number of observations of this _TYPE_ and _WAY_ in the data set	Total number of observations in the data set
0	0	0	0	0	Total	1	
0	0	1	1	1	A	a	1+a
0	1	0	1	2	B	b	
0	1	1	2	3	A*B	a*b	1+a+b+a*b
1	0	0	1	4	C	c	
1	0	1	2	5	A*C	a*c	
1	1	0	2	6	B*C	b*c	1+a+b+a*b+c
1	1	1	3	7	A*B*C	a*b*c	+a*c+b*c+a*b*c
Character binary equivalent of _TYPE_ (CHARTYPE option in the PROC HPSUMMARY statement)						a,b,c = number of levels of A, B, C, respectively	

Keywords and Formulas

Simple Statistics

The HPSUMMARY procedure uses a standardized set of keywords to refer to statistics. You specify these keywords in SAS statements to request the statistics to be displayed or stored in an output data set.

In the following notation, summation is over observations that contain nonmissing values of the analyzed variable and, except where shown, over nonmissing weights and frequencies of one or more:

x_i
: is the nonmissing value of the analyzed variable for observation i.

f_i
: is the frequency that is associated with x_i if you use a FREQ statement. If you omit the FREQ statement, then $f_i = 1$ for all i.

w_i
: is the weight that is associated with x_i if you use a WEIGHT statement. The HPSUMMARY procedure automatically excludes the values of x_i with missing weights from the analysis.

  By default, the HPSUMMARY procedure treats a negative weight as if it is equal to 0. However, if you use the EXCLNPWGT option in the PROC HPSUMMARY statement, then the procedure also excludes those values of with nonpositive weights.

  If you omit the WEIGHT statement, then $w_i = 1$ for all i.

n

is the number of nonmissing values of x_i, $\sum f_i$. If you use the EXCLNPWGT option and the WEIGHT statement, then n is the number of nonmissing values with positive weights.

$\bar{x}$

is the mean

$$\sum w_i x_i \Big/ \sum w_i$$

s^2

is the variance

$$\frac{1}{d} \sum w_i (x_i - \bar{x})^2$$

where d is the variance divisor (the VARDEF= option) that you specify in the PROC HPSUMMARY statement. Valid values are as follows:

When VARDEF=	*d* equals
N	n
DF	$n - 1$
WEIGHT | WGT	$\sum_i w_i$
WDF	$\left(\sum_i w_i\right) - 1$

The default is DF.

z_i

is the standardized variable

$$(x_i - \bar{x})/s$$

PROC HPSUMMARY calculates the following simple statistics:

- number of missing values
- number of nonmissing values
- number of observations
- sum of weights
- mean
- sum
- extreme values
- minimum
- maximum
- range

- uncorrected sum of squares
- corrected sum of squares
- variance
- standard deviation
- standard error of the mean
- coefficient of variation
- skewness
- kurtosis
- confidence limits of the mean
- median
- mode
- percentiles/deciles/quartiles
- *t* test for mean=0

The standard keywords and formulas for each statistic follow. Some formulas use keywords to designate the corresponding statistic.

Descriptive Statistics

The keywords for descriptive statistics are as follows:

CSS
: is the sum of squares corrected for the mean, computed as

$$\sum w_i (x_i - \bar{x})^2$$

CV
: is the percent coefficient of variation, computed as

$$(100s)/\bar{x}$$

KURTOSIS | KURT
: is the kurtosis, which measures heaviness of tails. When VARDEF=DF, the kurtosis is computed as

$$c_{4_n} \sum z_i^4 - \frac{3\,(n-1)^2}{(n-2)\,(n-3)}$$

where c_{4_n} is $\frac{n(n+1)}{(n-1)(n-2)(n-3)}$
When VARDEF=N, the kurtosis is computed as

$$\frac{1}{n} \sum z_i^4 - 3$$

The formula is invariant under the transformation $w_i^* = z w_i$, $z > 0$. When you use VARDEF=WDF or VARDEF=WEIGHT, the kurtosis is set to missing.

MAX
is the maximum value of x_i.

MEAN
is the arithmetic mean $\bar{x}$.

MIN
is the minimum value of x_i.

MODE
is the most frequent value of x_i.

NOTE: When QMETHOD=P2, PROC HPSUMMARY does not compute MODE.

N
is the number of x_i values that are not missing. Observations with $f_i < 1$ and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of N.

NMISS
is the number of x_i values that are missing. Observations with $f_i < 1$ and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of NMISS.

NOBS
is the total number of observations and is calculated as the sum of N and NMISS. However, if you use the WEIGHT statement, then NOBS is calculated as the sum of N, NMISS, and the number of observations excluded because of missing or nonpositive weights.

RANGE
is the range and is calculated as the difference between maximum value and minimum value.

SKEWNESS | SKEW
is skewness, which measures the tendency of the deviations to be larger in one direction than in the other. When VARDEF=DF, the skewness is computed as

$$c_{3_n} \sum z_i^3$$

where c_{3_n} is $\frac{n}{(n-1)(n-2)}$.
When VARDEF=N, the skewness is computed as

$$\frac{1}{n} \sum z_i^3$$

The formula is invariant under the transformation $w_i^* = z w_i$, $z > 0$. When you use VARDEF=WDF or VARDEF=WEIGHT, the skewness is set to missing.

STDDEV | STD
is the standard deviation s and is computed as the square root of the variance, s^2.

STDERR | STDMEAN
is the standard error of the mean, computed as

$$\frac{s}{\sqrt{\sum w_i}}$$

when VARDEF=DF, which is the default. Otherwise, STDERR is set to missing.

SUM
is the sum, computed as

$$\sum w_i x_i$$

SUMWGT
is the sum of the weights, W, computed as

$$\sum w_i$$

USS
is the uncorrected sum of squares, computed as

$$\sum w_i x_i^2$$

VAR
is the variance s^2.

Quantile and Related Statistics

The keywords for quantiles and related statistics are as follows:

MEDIAN
is the middle value.

P*n*
is the *n*th percentile. For example, P1 is the first percentile, P5 is the fifth percentile, P50 is the 50th percentile, and P99 is the 99th percentile.

Q1
is the lower quartile (25th percentile).

Q3
is the upper quartile (75th percentile).

QRANGE
is interquartile range and is calculated as

$$Q_3 - Q_1$$

You use the QNTLDEF= option to specify the method that the HPSUMMARY procedure uses to compute percentiles. Let n be the number of nonmissing values for a variable, and let $x_1, x_2, \ldots, x_n$ represent the ordered values of the variable such that x_1 is the smallest value, x_2 is the next smallest value, and x_n is the largest value. For the tth percentile between 0 and 1, let $p = t/100$. Then define j as the integer part of np and g as the fractional part of np or $(n+1)p$, so that

$$\begin{aligned} np &= j + g \quad \text{when QNTLDEF=1, 2, 3, or 5} \\ (n+1)p &= j + g \quad \text{when QNTLDEF=4} \end{aligned}$$

Here, QNTLDEF= specifies the method that the procedure uses to compute the tth percentile, as shown in Table 10.10.

When you use the WEIGHT statement, the tth percentile is computed as

$$y = \begin{cases} \frac{1}{2}(x_i + x_{i+1}) & \text{if } \sum_{j=1}^{i} w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^{i} w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

where w_j is the weight associated with x_i and $W = \sum_{i=1}^{n} w_i$ is the sum of the weights. When the observations have identical weights, the weighted percentiles are the same as the unweighted percentiles with QNTLDEF=5.

Table 10.10 Methods for Computing Quantile Statistics

QNTLDEF=	Description	Formula	
1	Weighted average at x_{np}	$y = (1-g)x_j + gx_{j+1}$ where x_0 is taken to be x_1	
2	Observation numbered closest to np	$y = x_i$	if $g \neq \frac{1}{2}$
		$y = x_j$	if $g = \frac{1}{2}$ and j is even
		$y = x_{j+1}$	if $g = \frac{1}{2}$ and j is odd
		where i is the integer part of $np + \frac{1}{2}$	
3	Empirical distribution function	$y = x_j$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$
4	Weighted average aimed at $x_{(n+1)p}$	$y = (1-g)x_j + gx_{j+1}$ where x_{n+1} is taken to be x_n	
5	Empirical distribution function with averaging	$y = \frac{1}{2}(x_j + x_{j+1})$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$

Hypothesis Testing Statistics

The keywords for hypothesis testing statistics are as follows:

T

is the Student's t statistic to test the null hypothesis that the population mean is equal to μ_0 and is calculated as

$$\frac{\bar{x} - \mu_0}{s \Big/ \sqrt{\sum w_i}}$$

By default, μ_0 is equal to zero. You must use VARDEF=DF, which is the default variance divisor; otherwise T is set to missing.

By default, when you use a WEIGHT statement, the procedure counts the x_i values with nonpositive weights in the degrees of freedom. Use the EXCLNPWGT option in the PROC HPSUMMARY statement to exclude values with nonpositive weights.

PROBT | PRT
: is the two-tailed p-value for the Student's t statistic, T, with $n-1$ degrees of freedom. This value is the probability under the null hypothesis of obtaining a more extreme value of T than is observed in this sample.

Confidence Limits for the Mean

The keywords for confidence limits are as follows:

CLM
: is the two-sided confidence limit for the mean. A two-sided $100(1-\alpha)$ percent confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1-\alpha/2;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

where s is $\sqrt{\frac{1}{n-1}\sum(x_i-\bar{x})^2}$, $t_{(1-\alpha/2;n-1)}$ is the $(1-\alpha/2)$ critical value of the Student's t statistic with $n-1$ degrees of freedom, and α is the value of the ALPHA= option which by default is 0.05. Unless you use VARDEF=DF (which is the default variance divisor), CLM is set to missing.

LCLM
: is the one-sided confidence limit below the mean. The one-sided $100(1-\alpha)$ percent confidence interval for the mean has the lower limit

$$\bar{x} - t_{(1-\alpha;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF (which is the default variance divisor), LCLM is set to missing.

UCLM
: is the one-sided confidence limit above the mean. The one-sided $100(1-\alpha)$ percent confidence interval for the mean has the upper limit

$$\bar{x} + t_{(1-\alpha;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF (which is the default variance divisor), UCLM is set to missing.

Data Requirements for the HPSUMMARY Procedure

The following are the minimal data requirements to compute unweighted statistics and do not describe recommended sample sizes. Statistics are reported as missing if VARDEF=DF (the default) and the following requirements are not met:

- N and NMISS are computed regardless of the number of missing or nonmissing observations.
- SUM, MEAN, MAX, MIN, RANGE, USS, and CSS require at least one nonmissing observation.

- VAR, STD, STDERR, CV, T, PRT, and PROBT require at least two nonmissing observations.
- SKEWNESS requires at least three nonmissing observations.
- KURTOSIS requires at least four nonmissing observations.
- SKEWNESS, KURTOSIS, T, PROBT, and PRT require that STD is greater than zero.
- CV requires that MEAN is not equal to zero.
- CLM, LCLM, UCLM, STDERR, T, PRT, and PROBT require that VARDEF=DF.

References

Jain, R. and Chlamtac, I. (1985), "The P^2 Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations," *Communications of the ACM*, 28, 1076–1085.

Subject Index

Syntax Index