



Base SAS[®] Procedures Guide High-Performance Procedures Sixth Edition

2020.1*

* This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2020. *Base SAS® Procedures Guide: High-Performance Procedures*. Cary, NC: SAS Institute Inc.

Base SAS® Procedures Guide: High-Performance Procedures

Copyright © 2020, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

March 2023

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

SAS software may be provided with certain third-party software, including but not limited to open source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with SAS software, refer to [Third-Party Software Reference | SAS Support](#).

v_001-P1:prochp

Contents

Chapter 1.	Introduction	1
Chapter 2.	Shared Concepts and Topics	5
Chapter 3.	The HPBIN Procedure	7
Chapter 4.	The HPCORR Procedure	27
Chapter 5.	The HPDMDB Procedure	43
Chapter 6.	The HPDS2 Procedure	53
Chapter 7.	The HPIMPUTE Procedure	67
Chapter 8.	The HPSAMPLE Procedure	75
Chapter 9.	The HPSUMMARY Procedure	85

Chapter 1

Introduction

Contents

Overview of Base SAS High-Performance Procedures	1
About This Book	2
Chapter Organization	2
Typographical Conventions	2
Options Used in Examples	3
SAS Technical Support Services	3

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 2, “[Shared Concepts and Topics](#).”

About This Book

This book assumes that you are familiar with Base SAS software and with the books *SAS Programmers Guide: Essentials* 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 1, this chapter, provides an overview of high-performance utility procedures.

Chapter 2, “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 functionality 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.
<i>oblique</i>	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.

<i>italic</i>	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.

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 2

Shared Concepts and Topics

Contents

Overview	5
PERFORMANCE Statement	5
CODE Statement	6

Overview

This chapter describes some syntax that is common to high-performance analytical procedures.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > ;

The PERFORMANCE statement defines performance parameters for multithreaded computing and requests detailed results about the performance characteristics of a high-performance analytical procedure.

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

BPC=*n*

specifies the number of bytes per character that is used in processing character strings in multibyte encodings. The default is the bytes per character of the encoding. The number of characters in a string is calculated as the byte length of the string divided by the bytes per character of the encoding. This will be incorrect when the strings in the multibyte encoding contain one or more single byte characters. In such cases, setting BPC=1 enables appropriate byte lengths to be used in processing such strings.

DETAILS

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

NTHREADS=*n*

THREADS=*n*

specifies the number of threads for analytic computations and overrides the SAS system option THREADS | NOTTHREADS. 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.

CODE Statement

CODE < options > ;

The `CODE` statement enables you to write SAS DATA step code to a file or catalog entry for computing predicted values of the fitted model. This code can then be included in a DATA step to score new data. For example, in the following program, the `CODE` statement writes the code for predicting the outcome of a logistic regression model to the file *mycode.sas*. The file is subsequently included in a DATA step to score the `sashelp.Bmt` data.

```
proc hplogistic data=sashelp.Bmt;
  class Group;
  model Status=Group;
  code file='mycode.sas';
run;

data Score;
  set sashelp.Bmt;
  %include mycode;
run;
```

This statement is supported by several procedures. The *options* you can specify in the `CODE` statement are described in each procedure that supports this statement.

Chapter 3

The HPBIN Procedure

Contents

Overview: HPBIN Procedure	8
Bucket Binning	8
Winsorized Binning	8
Quantile Binning	8
Pseudo-Quantile Binning	8
PROC HPBIN Features	9
Getting Started: HPBIN Procedure	9
Syntax: HPBIN Procedure	11
PROC HPBIN Statement	12
CODE Statement	14
FREQ Statement	14
ID Statement	14
INPUT Statement	14
PERFORMANCE Statement	15
TARGET Statement	15
Details: HPBIN Procedure	16
Computing the Quantiles (Percentiles)	16
Binning Computation and Formulas	16
Computing the Weight of Evidence and Information Value	18
Data Output	19
ODS Tables	20
Examples	21
Example 3.1: Bucket Binning	21
Example 3.2: Winsorized Binning	24
Example 3.3: Bucket Binning and Weight-of-Evidence Computation	25

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.

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 16.

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 exploits all the available cores and concurrent threads.

Getting Started: HPBIN Procedure

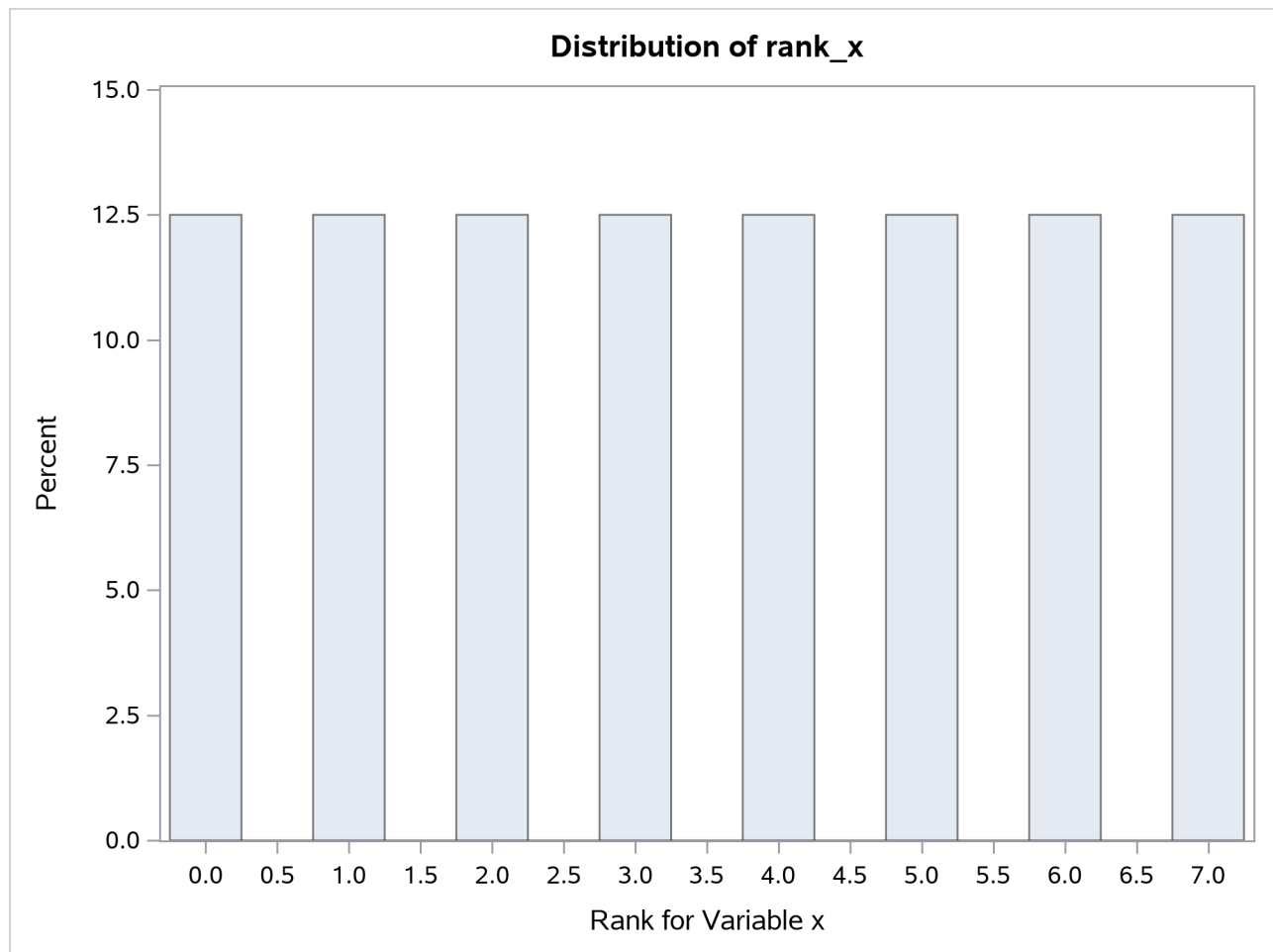
This example shows how you can use the HPBIN procedure to perform pseudo-quantile binning. Consider the following statements:

```
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 3.1](#).

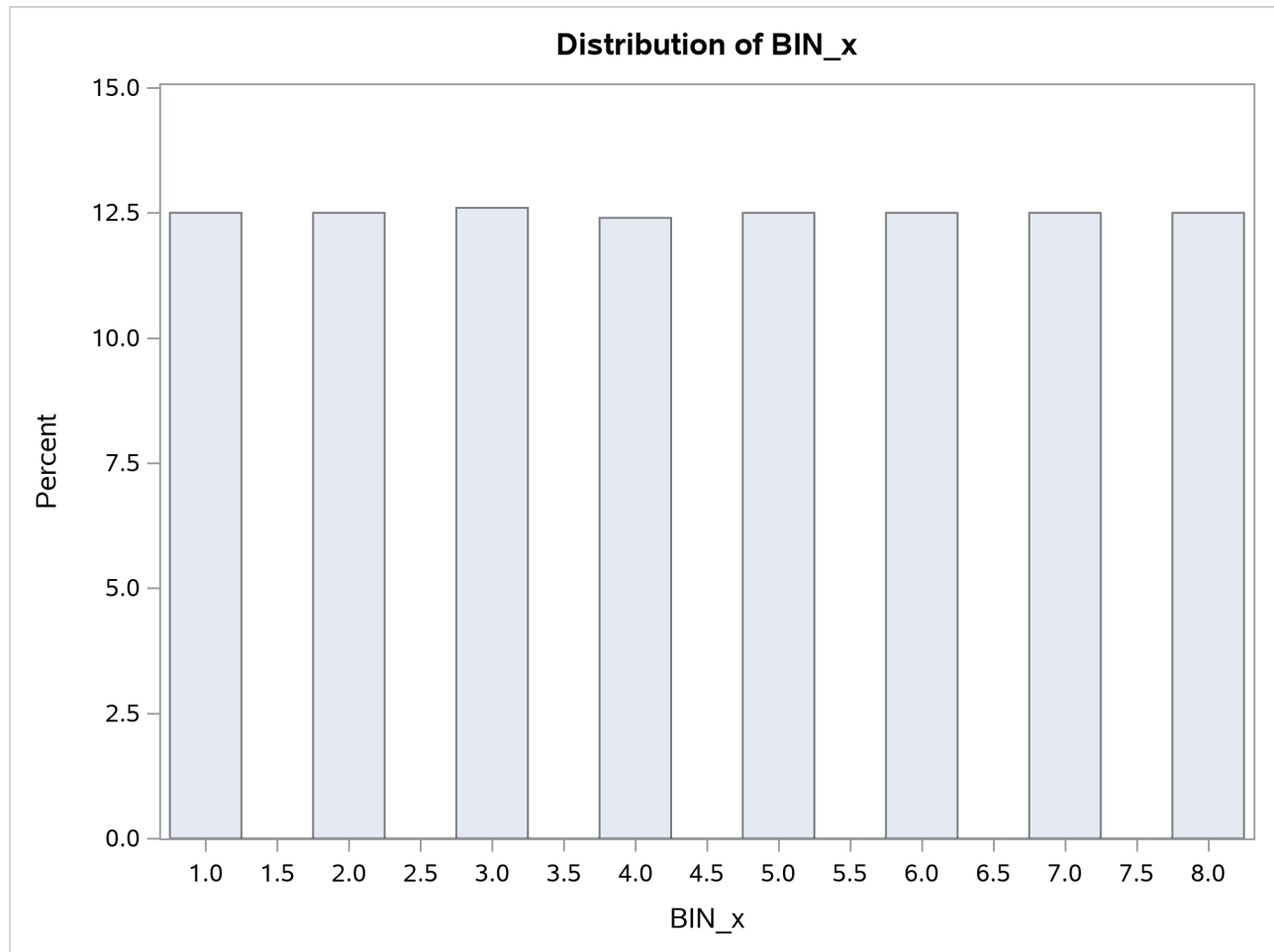
Figure 3.1 Histogram for Rank X

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.

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 3.2](#).) This histogram is similar to the one in [Figure 3.1](#).

```
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 3.2 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 3.1 summarizes important *options* in the PROC HPBIN statement by function.

Table 3.1 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 to be used by PROC HPBIN.

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—for example, if the binning method is QUANTILE and the specified binning level is greater than the sample size. In this case, PROC HPBIN issues 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. The output data set 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 issues 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—for example, if the binning method is QUANTILE and the

specified binning level is greater than the sample size. In this case, PROC HPBIN issues 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 computing and requests detailed results about the performance characteristics of the HPBIN procedure.

The PERFORMANCE statement is documented further in the section “[PERFORMANCE Statement](#)” on page 5 of Chapter 2, “[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 $x_1, x_2, \dots, x_m$ represent the ordered values of the variable. Let the t th 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 t th 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 $i = 1, 2, \dots, n$. 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 $k = 1, 2, \dots, \text{numbin} - 1$, and numbin is the value of the NUMBIN= option in the PROC HPBIN statement.

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

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

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. 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 16.

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 $B_i, i = 1, 2, \dots, N$, 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 $p_k \in P, k = 1, 2, \dots, 11$, 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 $k = 1, 2, \dots, 11$.

PROC HPBIN calculates the split points as follows:

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

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

$$\text{and} \quad \sum_{i=1}^{I_k} c_i > \sum_{i=1}^{I_{k-1}} c_i$$

$$\text{and} \quad \sum_{i=1}^{I_k} c_i < n$$

where k is the k th split.

The split value is

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

where $k = 1, 2, \dots$, and $k < \text{numbin}$.

The time complexity for pseudo-quantile binning is $C\mathcal{O}(n)/(\text{nodes} * \text{cpus})$, 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 $\sum_{i=1}^{I_l} c_i > \text{lwc}$. 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

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

The trimmed mean is calculated by the formula

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

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:

$$\text{WOE}_{\text{attribute}} = \ln \frac{P_{\text{attribute}}^{\text{non-event}}}{P_{\text{attribute}}^{\text{event}}} = \ln \frac{\frac{N_{\text{attribute non-event}}}{N_{\text{non-event}}^{\text{total}}}}{\frac{N_{\text{attribute event}}}{N_{\text{event}}^{\text{total}}}}$$

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

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

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

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

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:

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

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 to a SAS data set. You can also take advantage of multithreading, which performs parallel computation.

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 3.2](#).

Table 3.2 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	The “Performance Information” table is produced by default. The table displays the number of threads used.

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 3.1: Bucket Binning

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 3.1.1](#) through [Output 3.1.2](#).

Output 3.1.1 PROC HPBIN Binning Information Display

The HPBIN Procedure

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

Output 3.1.2 PROC HPBIN Mapping Table

Mapping				
Variable	Binned Variable	Range	Frequency	Proportion
x1	BIN_x1	x1 < 0.0999254969	108	0.10800000
		0.0999254969 <= x1 < 0.1993986319	94	0.09400000
		0.1993986319 <= x1 < 0.2988717668	90	0.09000000
		0.2988717668 <= x1 < 0.3983449018	95	0.09500000
		0.3983449018 <= x1 < 0.4978180367	90	0.09000000
		0.4978180367 <= x1 < 0.5972911716	115	0.11500000
		0.5972911716 <= x1 < 0.6967643066	116	0.11600000
		0.6967643066 <= x1 < 0.7962374415	105	0.10500000
		0.7962374415 <= x1 < 0.8957105765	106	0.10600000
		0.8957105765 <= x1	81	0.08100000
x2	BIN_x2	x2 < 1.0215392099	99	0.09900000
		1.0215392099 <= x2 < 2.0154070426	110	0.11000000
		2.0154070426 <= x2 < 3.0092748753	99	0.09900000
		3.0092748753 <= x2 < 4.003142708	85	0.08500000
		4.003142708 <= x2 < 4.9970105407	109	0.10900000
		4.9970105407 <= x2 < 5.9908783734	98	0.09800000
		5.9908783734 <= x2 < 6.9847462061	100	0.10000000
		6.9847462061 <= x2 < 7.9786140388	94	0.09400000
		7.9786140388 <= x2 < 8.9724818715	112	0.11200000
		8.9724818715 <= x2	94	0.09400000
x3	BIN_x3	x3 < 10.173596707	97	0.09700000
		10.173596707 <= x3 < 20.14299647	97	0.09700000
		20.14299647 <= x3 < 30.112396232	106	0.10600000
		30.112396232 <= x3 < 40.081795994	94	0.09400000
		40.081795994 <= x3 < 50.051195757	102	0.10200000
		50.051195757 <= x3 < 60.020595519	99	0.09900000
		60.020595519 <= x3 < 69.989995281	107	0.10700000
		69.989995281 <= x3 < 79.959395044	92	0.09200000
		79.959395044 <= x3 < 89.928794806	91	0.09100000
		89.928794806 <= 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 3.1.3](#).

Output 3.1.3 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 3.2: 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;

proc hpbin data=ex12 NUMBIN=10 WINSOR WINSORRATE=0.05;
  input x1-x2;
run;
```

The preceding statements generate [Output 3.2.1](#) and [Output 3.2.2](#).

Output 3.2.1 PROC HPBIN Winsorized Statistics

The HPBIN Procedure

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

Output 3.2.2 PROC HPBIN Trimmed Statistics

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

Example 3.3: 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 c11 $ x0 x1 x2 y $ freq id;
  datalines;
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
;

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

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

Output 3.3.1 Number of Observations Table**The HPBIN Procedure**

Number of Observations Read	10
Number of Observations Used	7

Output 3.3.2 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 < 1.5	1	1	0	0	0.40546511	0.06757752
	1.5 <= x1 < 3	0	0	0	0	0	0
	3 <= x1 < 4.5	1	0.33333333	2	0.66666667	-1.3862944	0.69314718
	4.5 <= x1	2	0.66666667	1	0.33333333	0	0
x2	BIN_x2	0	0	0	0	0	0
	x2 < 2.2	2	1	0	0	0.91629073	0.30543024
	2.2 <= x2 < 3.4	0	0	0	0	0	0
	3.4 <= x2 < 4.6	2	0.66666667	1	0.33333333	0	0
	4.6 <= x2 < 5.8	0	0	0	0	0	0
	5.8 <= x2	2	0.50000000	2	0.50000000	-0.6931472	0.23104906

Output 3.3.3 Variable Information Value Table

Variable Information Value	
Variable	Information Value
x1	1.06615494
x2	0.53647930

Chapter 4

The HPCORR Procedure

Contents

Overview: HPCORR Procedure	28
PROC HPCORR Features	28
Getting Started: HPCORR Procedure	29
Syntax: HPCORR Procedure	31
PROC HPCORR Statement	31
FREQ Statement	34
PERFORMANCE Statement	34
VAR Statement	34
WEIGHT Statement	34
WITH Statement	35
Details: HPCORR Procedure	35
Pearson Product-Moment Correlation	35
Multithreading	36
Output Tables	37
Output Data Sets	38
ODS Table Names	39
Examples: HPCORR Procedure	39
Example 4.1: Computing the Pearson Measure of Association	39
References	40

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 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.

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 4.1](#) 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 4.1](#) displays univariate statistics for the analysis variables.

Figure 4.1 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 4.2](#) 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 4.2 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 4.2](#) 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 4.1 summarizes the *options* available in the PROC HPCORR statement.

Table 4.1 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

Table 4.1 *continued*

Option	Description
Printed Output	
BEST=	Displays the specified number of ordered correlation coefficients
NOCORR	Suppresses Pearson correlations
NOPRINT	Suppresses all printed output
NOPROB	Suppresses <i>p</i> -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 4.2 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 j th nonmissing observation.

Table 4.2 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 computing and requests detailed results about the performance characteristics of the HPCORR procedure.

The PERFORMANCE statement is documented further in the section “[PERFORMANCE Statement](#)” on page 5.

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 $X_1, \dots, X_m$ are analysis variables that you specify in the VAR statement and $Y_1, \dots, Y_n$ 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

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

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 and $\bar{y}$ 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 cpubcount=4;
```

- You can specify the `NTHREADS=` option in the `PERFORMANCE` statement to determine the number of threads. This specification overrides the `NOTHEADS` 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:

- `_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
- CORR, 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 4.3](#). You must use these names to refer to tables when you use the Output Delivery System (ODS). For more information about ODS, see Chapter 23, “Using the Output Delivery System” (*SAS/STAT User’s Guide*).

Table 4.3 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 4.1: Computing the Pearson Measure of Association

The Fitness data set created in the section “Getting Started: HPCORR Procedure” on page 29 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 4.1.1](#) 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 4.1.1 Simple Statistics

Measures of Association for a Physical Fitness Study

The HPCORR Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Output 4.1.1 *continued***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 4.1.2](#) 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 4.1.2 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.

References

- Anderson, T. W. (1984). *An Introduction to Multivariate Statistical Analysis*. 2nd ed. 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 ed. 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 *Encyclopedia of Statistical Sciences*, vol. 7, edited by S. Kotz, N. L. Johnson, and C. B. Read. 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 ed., revised and enlarged. Darien, 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 ed. 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 ed. New York: W. H. Freeman.
- Mudholkar, G. S. (1983). "Fisher's z-Transformation." In *Encyclopedia of Statistical Sciences*, vol. 3, edited by S. Kotz and N. L. Johnson. New York: John Wiley & Sons.
- 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 ed. New York: McGraw-Hill.
- Olsson, U., Drasgow, F., and Dorans, N. J. (1982). "The Polyserial Correlation Coefficient." *Psychometrika* 47:337–347.
- Ott, R. L., and Longnecker, M. T. (2000). *An Introduction to Statistical Methods and Data Analysis*. 5th ed. 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. <http://www2.sas.com/proceedings/sugi26/p246-26.pdf>.

Chapter 5

The HPDMDB Procedure

Contents

Overview: HPDMDB Procedure	43
Getting Started: HPDMDB Procedure	44
Syntax: HPDMDB Procedure	47
PROC HPDMDB Statement	47
CLASS Statement	48
FREQ Statement	49
PERFORMANCE Statement	49
VAR Statement	49
WEIGHT Statement	50
Details: HPDMDB Procedure	50
Examples: HPDMDB Procedure	51
Example 5.1: Running PROC HPDMDB on the Client	51

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.

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 single-machine mode.

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

- 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

Getting Started: HPDMDDB Procedure

The HPDMDDB 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 5.1](#)) contains the class summary table with levels sorted according to the sort option in the `CLASS` statement of `PROC HPDMDDB`. 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 5.1 Summaries of Classification Variables in `Sampsio.Hmeq` Data Set

The HPDMDDB 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 5.1 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 5.2.

Figure 5.2 Summaries of Numeric Variables in Sampsis.Hmeq Data Set

The HPDMDb Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Figure 5.2 continued

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

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 5.1](#) shows the possible values for *divisor*.

Table 5.1 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 HPDMDDB 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 computing, and requests detailed results about the performance characteristics of a SAS high-performance analytical procedure.

The PERFORMANCE statement for SAS high-performance analytical procedures is documented in the section “[PERFORMANCE Statement](#)” on page 5 of Chapter 2, “[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 statistics in [Table 5.2](#).

Table 5.2 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

Table 5.2 *continued*

Statistic	Meaning
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 49.

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 **MyLevel1** 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: HPDMDDB Procedure

Example 5.1: Running PROC HPDMDDB on the Client

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

```
data ex;
input x1 x2 x3 x4 y$ w f y2;
datalines;
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 5.1.1 shows the summaries of the numeric variables in the data set ex.

Output 5.1.1 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 5.1.2 shows the summaries of the classification variables in the data set ex.

Output 5.1.2 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

The results are the same as those shown in [Output 5.1.1](#) and [Output 5.1.2](#).

Chapter 6

The HPDS2 Procedure

Contents

Overview: HPDS2 Procedure	54
PROC HPDS2 Features	54
Getting Started: HPDS2 Procedure	55
Syntax: HPDS2 Procedure	57
PROC HPDS2 Statement	57
DATA Statement	57
ENDDATA Statement	58
PERFORMANCE Statement	58
QUIT Statement	58
RUN Statement	58
RUN CANCEL Statement	59
Details: HPDS2 Procedure	59
Parallel Execution of DS2 Code	59
Limitations and Issues	59
Packages	59
Threads	60
PERFORMANCE Statement Options	60
Data Input/Output	60
Data Types and Declarations	60
Error Messages	60
Examples: HPDS2 Procedure	61
Example 6.1: Compute Mandelbrot Set	61
Example 6.2: Aggregate Result Data Set	63

Overview: HPDS2 Procedure

The HPDS2 procedure enables you to submit DS2 language statements from a Base SAS session for execution in a number of parallel threads on the local client machine.

Because the DS2 code is executed in parallel in separate threads, each thread produces separate output that is the result of processing only a subset of the data. As a result, it might be necessary to use a second-stage program to aggregate the results from each thread. The second stage can be executed by using the DS2 procedure, where the SET statement reads all rows created by the preceding parallel stage.

The syntax of the DS2 language is similar to the syntax of the DATA step, but it does not include several key statements such as INPUT. In addition, using DS2 along with SAS high-performance analytical procedures limits the 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. The HPDS2 procedure 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*.

PROC HPDS2 Features

The following list summarizes the basic features of the HPDS2 procedure:

- provides the ability to execute one or more copies of a DS2 program in parallel in multiple threads on the local client machine
- enables control of the level of parallel threads used for execution
- exploits all the available cores and concurrent threads
- performs a syntax check of the DS2 code before submitting it for parallel execution

Getting Started: HPDS2 Procedure

This example illustrates a simple HPDS2 procedure. The DS2 code that is submitted for execution 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
dtw 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 output data set, and the SET DS2GTF.in statement refers to the input data set:

```
proc hpds2 data=daily_temps
  out=avg_temps;
  performance nthreads=4;
  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=avg_temps;
  title1 'Average Temperatures';
  var city synopsis avgf avgc;
run;

```

Figure 6.1 displays the output of the PRINT procedure.

Figure 6.1 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	dtw	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 data set to be used by PROC HPDS2. The default is the most recently created data set.

OUTPUT=data-set

OUT=data-set

names the data set 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 output and input data sets, respectively.

ENDDATA Statement

ENDDATA ;

The ENDDATA statement terminates the DS2 code block. The statements between the DATA and ENDDATA statement are submitted for parallel 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 and requests results about the performance characteristics of PROC HPDS2.

The NTHREADS= option determines how many independent copies of the DS2 program are run in parallel. Setting the NTHREADS= option to a value that is greater than the CPU count is not likely to improve overall throughput.

For more information about the PERFORMANCE statement, see the section “[PERFORMANCE Statement](#)” on page 5 of Chapter 2, “[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

The HPDS2 procedure controls the number of parallel threads used for 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.

By default, the number of copies of the DS2 program that are executed in parallel 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.

An important characteristic of multithreaded applications is that they might produce nondeterministic or unpredictable results. 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 thread. 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 the HPDS2 procedure. Not all DS2 language elements can be meaningfully used in multithreaded applications. For example, lagging or retaining of variables can imply ordering of observations. A deterministic order of observations does not exist in this context.

Limitations and Issues

The current release of the HPDS2 procedure does not support all of the features of the DS2 language. The following subsections summarize 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 statements in the HPDS2 procedure results in an error.

Threads

The HPDS2 procedure does not support the DS2 thread syntax. Use of the `THREAD` and `ENDTHREAD` statements in the 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 the HPDS2 procedure is not supported at this time.

The use of nested SQL within the `SET` statement of the HPDS2 procedure is not supported at this time.

When used in the 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 in the 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 in the HPDS2 procedure.

Error Messages

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

Examples: HPDS2 Procedure

Example 6.1: 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 for parallel execution by the HPDS2 procedure. 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.

```

/* 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=mandelbrot;
  performance nthreads=4;
  data DS2GTF.out;
    dcl int mesh;
    dcl double x y rr nx ny;
    keep p q mesh;
  method run();
    set DS2GTF.in;
    x = p;

```

```

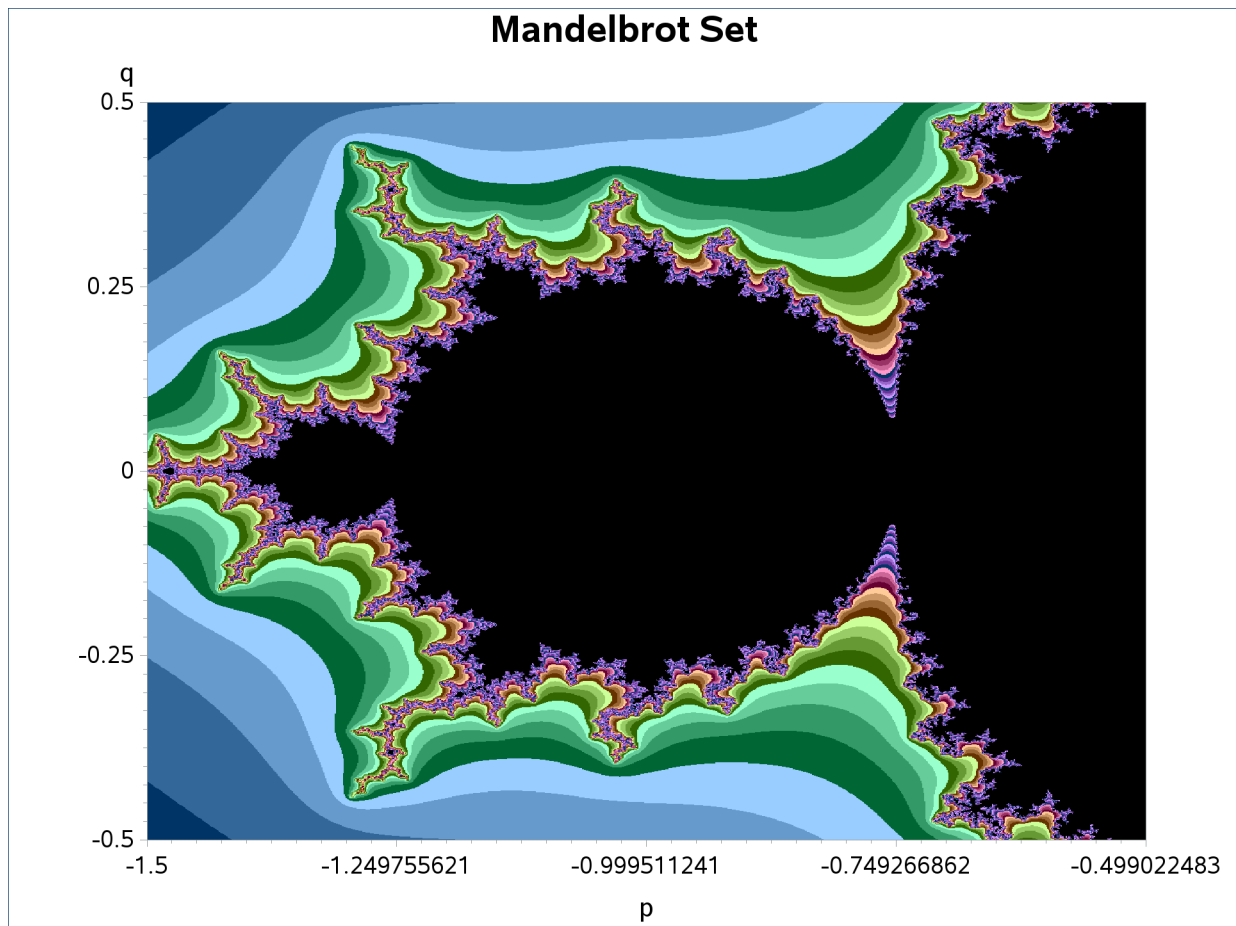
        y = q;
        rr = r**2;
        mesh = 0;
        do while (mesh < maxiterate and (x**2+y**2 < rr));
            nx = x**2 - y**2 + p;
            ny = 2.0*x*y + q;
            x = nx;
            y = ny;
            mesh = mesh+1;
        end;
    end;
enddata;
run;

/* 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=mandelbrot;
    Title 'Mandelbrot Set';
    plot q*p=mesh /
        nolegend
        pattern
        join
        levels = 5 to 45
    ;
run;

```

Output 6.1.1 shows the graphic representation of the Mandelbrot set that is computed by the HPDS2 procedure.

Output 6.1.1 Computed Mandelbrot Set

Example 6.2: Aggregate Result Data Set

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

This example uses a DATA step program 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 with the HPDS2 procedure 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.

```
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=obj_comps;
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=obj_comps (obs=20);
    title1 'Volumes and Areas';
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 6.2.1 shows a subset of the volumes and areas that are computed by the HPDS2 procedure.

Output 6.2.1 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 6.2.2 shows the aggregated results that are produced by the second-stage DS2 program.

Output 6.2.2 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 7

The HPIMPUTE Procedure

Contents

Overview: HPIMPUTE Procedure	67
PROC HPIMPUTE Features	68
Getting Started: HPIMPUTE Procedure	68
Syntax: HPIMPUTE Procedure	69
PROC HPIMPUTE Statement	69
CODE Statement	70
FREQ Statement	70
ID Statement	70
IMPUTE Statement	70
INPUT Statement	71
PERFORMANCE Statement	71
Details: HPIMPUTE Procedure	72
Obtaining the Statistics for Imputation	72
Random Imputation	73
Displayed Output	73
ODS Table Names	73
Examples: HPIMPUTE Procedure	74
Example 7.1: Running Client Data on the Client	74

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.

PROC HPIMPUTE Features

The following list summarizes the basic features of PROC HPIMPUTE:

- provides the ability to execute imputation
- exploits all the available concurrent threads

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;
DATALINES;
  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 7.1 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 7.1 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.

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. You can specify the following *option*:

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 computing and requests detailed results about the performance characteristics of the HPIMPUTE procedure.

The PERFORMANCE statement is documented further in the section “[PERFORMANCE Statement](#)” on page 5 of Chapter 2, “[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. 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 $i = 1, 2, \dots, n$. 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 $\text{range}(x) = \max(x) - \min(x)$.

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 , $i = 1, 2, \dots, N$, 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 $\sum_{i=1}^I c_i \geq 0.5 * m$, 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 $i = 1, 2, \dots, n$. Let $\min(x) = \min_{i \in \{1..n\}} \{x_i\}$, and let $\max(x) = \max_{i \in \{1..n\}} \{x_i\}$. The random value $R = \min(x) + (\max(x) - \min(x)) * \text{ranuni}(\text{SEED})$, where $\text{ranuni}()$ is a function that takes a `SEED` (number) as input and returns a random value from a uniform distribution between 0 and 1. The `SEED` is set to 5 by default.

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. The table displays the number of threads used.

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 7.1 lists the names of the data tables that are created by the `HPIMPUTE` procedure. You must use these names in ODS statements.

Table 7.1 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 7.1: Running Client Data on the Client

This example demonstrates how to use PROC HPIMPUTE to perform imputation on the Sampsio.Hmeq data set.

```
/*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 7.1.1 shows the imputation results.

Output 7.1.1 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 7.1.2 shows the performance information.

Output 7.1.2 Performance Information
The HPIMPUTE Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Chapter 8

The HPSAMPLE Procedure

Contents

Overview: HPSAMPLE Procedure	75
Getting Started: HPSAMPLE Procedure	76
Syntax: HPSAMPLE Procedure	77
PROC HPSAMPLE Statement	77
CLASS Statement	79
PERFORMANCE Statement	79
TARGET Statement	79
VAR Statement	80
Details: HPSAMPLE Procedure	80
Class Level	80
Displayed Output	80
ODS Table Names	81
Examples: HPSAMPLE Procedure	81
Example 8.1: Using PROC HPSAMPLE to Perform Simple Random Sampling	81
Example 8.2: Using PROC HPSAMPLE to Perform Stratified Sampling	82
Example 8.3: Using PROC HPSAMPLE to Perform Oversampling	84

Overview: HPSAMPLE Procedure

The HPSAMPLE procedure is a high-performance procedure that performs simple random sampling, stratified sampling, or oversampling. This procedure runs in single-machine 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

The HPSAMPLE procedure does the following:

- 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 PROC HPSAMPLE uses only a single thread)

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 8.1) that shows the performance environment information and a frequency table (Figure 8.2) that shows the frequency of observations in each level of BAD.

Figure 8.1 Performance Information

The HPSAMPLE Procedure	
Performance Information	
Execution Mode	Single-Machine
Number of Threads	1

Figure 8.2 Frequency Table

One Target Stratified Sampling Frequency Table		
Target	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 statement 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.

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 in single machine mode.

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 variables specified in the CLASS statement.

PERFORMANCE Statement

PERFORMANCE < *performance-options* > ;

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

The PERFORMANCE statement for SAS high-performance analytical procedures is documented in the section “[PERFORMANCE Statement](#)” on page 5 of Chapter 2, “[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 **MyLevel1** 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**.

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 the number of threads used.

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 8.1](#).

Table 8.1 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 8.1: Using PROC HPSAMPLE to Perform Simple Random Sampling

This example demonstrates how to use PROC HPSAMPLE to perform simple random sampling on the Sampsio.Hmeq data set.

```
/*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 8.1.1 shows the number of observations in the data set sampsio.hmeq and the number of samples.

Output 8.1.1 Frequency Table

The HPSAMPLE Procedure

Simple Random Sampling Frequency Table	
Number of Obs	Number of Samples
5960	22

Output 8.1.2 shows the sample data.

Output 8.1.2 Sample Data

Obs	JOB	REASON	LOAN	VALUE	DELINQ	DEROG
1	Other	Homelmp	4900	65774	2	1
2	Other	Homelmp	5700	82923	0	0
3	Other	Homelmp	7000	124827	.	.
4		DebtCon	8300	75081	0	0
5	Office	DebtCon	10000	125500	0	0
6	Other		11100	61406	0	0
7	Other	Homelmp	11500	64037	0	0
8	ProfExe	Homelmp	11900	105454	0	0
9	Mgr	Homelmp	15000	122400	2	0
10	Other	Homelmp	15000	107207	0	0
11	ProfExe	Homelmp	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	Homelmp	21400	103427	0	0
17	Mgr	Homelmp	22400	121601	0	0
18	Other	Homelmp	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 8.2: Using PROC HPSAMPLE to Perform Stratified Sampling

This example uses the same data set as is used in [Example 8.1](#). This example demonstrates how to use PROC HPSAMPLE to perform stratified sampling.

```
proc hpsample data=sampsio.hmeq out=out2 sampct=10 seed=13579 partition;
    var loan value delinq derog;
    class job reason;
    target job;
run;
proc print data=out2(obs=15);
run;
```

[Output 8.2.1](#) shows the frequency information for each level of target variable job in the data set Sampsio.Hmeq and in the sample.

Output 8.2.1 Frequency Table**The HPSAMPLE Procedure**

One Target Stratified Sampling Frequency Table		
Target	Number of Obs	Number of Samples
	279	28
MGR	767	76
OFFICE	948	94
OTHER	2388	238
PROFEXE	1276	128
SALES	109	11
SELF	193	19

Output 8.2.2 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 8.2.2 Sample Output with Partition Indicator

Obs	JOB	REASON	LOAN	VALUE	DELINQ	DEROG	_Partind_
1	Other	Homelmp	1100	39025	0	0	0
2	Other	Homelmp	1300	68400	2	0	0
3	Other	Homelmp	1500	16700	0	0	0
4			1500	.	.	.	0
5	Office	Homelmp	1700	112000	0	0	0
6	Other	Homelmp	1700	40320	0	0	0
7	Other	Homelmp	1800	57037	2	3	0
8	Other	Homelmp	1800	43034	0	0	0
9	Other	Homelmp	2000	46740	2	0	0
10	Sales	Homelmp	2000	62250	0	0	0
11			2000	.	.	.	0
12	Office	Homelmp	2000	29800	1	0	0
13	Other	Homelmp	2000	55000	0	0	0
14	Mgr		2000	87400	0	0	0
15	Other	Homelmp	2100	83850	1	0	0

Example 8.3: Using PROC HPSAMPLE to Perform Oversampling

This example uses the same data set as in [Example 8.1](#). It demonstrates how to use PROC HPSAMPLE to perform oversampling.

The `ods output FreqTable=Freqtab;` statement saves the frequency table to a SAS data set called `Freqtab` on the client.

```
proc hpsample data=sampsio.hmeq out=out3 seed=13579 partition
  sampctevt=80 eventprop=.2 event="SALES";
  var loan value delinq derog;
  class job;
  target job;
  ods output FreqTable=Freqtab;
run;
```

[Output 8.3.1](#) shows the number of observations in each level of target variable `job` in the data set `Sampsio.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 8.3.1 Frequency Table
The HPSAMPLE Procedure

Oversampling Frequency Table		
Target	Number of Obs	Number of Samples
	279	16
MGR	767	46
OFFICE	948	57
OTHER	2388	142
PROFEXE	1276	76
SALES	109	87
SELF	193	12

Chapter 9

The HPSUMMARY Procedure

Contents

Overview: HPSUMMARY Procedure	86
PROC HPSUMMARY Features	86
PROC HPSUMMARY Compared with Other Procedures	87
Getting Started: HPSUMMARY Procedure	87
Syntax: HPSUMMARY Procedure	89
PROC HPSUMMARY Statement	89
Statistic Keywords	92
CLASS Statement	94
FREQ Statement	95
OUTPUT Statement	95
PERFORMANCE Statement	97
TYPES Statement	97
VAR Statement	98
WAYS Statement	99
WEIGHT Statement	100
Details: HPSUMMARY Procedure	101
How PROC HPSUMMARY Groups Data	101
Computational Resources	102
Statistical Computations	103
Results	105
Keywords and Formulas	107
References	114

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 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 in-memory summarization within a multiprocessor or multicore computer. The following list summarizes the basic features of PROC HPSUMMARY:

- performs all computations in memory, avoiding the explicit use of intermediate storage
- provides the ability to execute summarization in parallel
- is highly multithreaded
- enables you to control the level of parallelism

PROC HPSUMMARY Compared 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 the types.

Getting Started: HPSUMMARY Procedure

This example illustrates a simple use of the HPSUMMARY procedure to summarize a simple data set. The following DATA step creates a data set that consists of test scores:

```
data 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:

```
proc hpsummary data=grades;
  var Score;
  class Status Year;
  types () status*year;
  output out=result;
run;
proc print data=result;
run;
```

Figure 9.1 displays the tables produced by the HPSUMMARY procedure.

Figure 9.1 HPSUMMARY Output
The HPSUMMARY Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Figure 9.1 *continued*

Data Access Information						
Data	Engine	Role	Path			
WORK.GRADES	V9	Input	On Client			
WORK.RESULT	V9	Output	On Client			

Obs	Status	Year	_TYPE_	_FREQ_	_STAT_	Score
1			0	10	N	10.0000
2			0	10	MIN	78.0000
3			0	10	MAX	92.0000
4			0	10	MEAN	86.0000
5			0	10	STD	4.7140
6	1	97	3	3	N	3.0000
7	1	97	3	3	MIN	78.0000
8	1	97	3	3	MAX	91.0000
9	1	97	3	3	MEAN	84.6667
10	1	97	3	3	STD	6.5064
11	1	98	3	3	N	3.0000
12	1	98	3	3	MIN	84.0000
13	1	98	3	3	MAX	92.0000
14	1	98	3	3	MEAN	88.3333
15	1	98	3	3	STD	4.0415
16	2	97	3	3	N	3.0000
17	2	97	3	3	MIN	82.0000
18	2	97	3	3	MAX	90.0000
19	2	97	3	3	MEAN	86.6667
20	2	97	3	3	STD	4.1633
21	2	98	3	1	N	1.0000
22	2	98	3	1	MIN	81.0000
23	2	98	3	1	MAX	81.0000
24	2	98	3	1	MEAN	81.0000
25	2	98	3	1	STD	.

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 DATA Step 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 9.1 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 90. For information about the *statistic-keywords*, see the section “[Statistic Keywords](#)” on page 92.

Table 9.1 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 <i>n</i> -way)
CHARTYPE	Specifies that the _TYPE_ variable contain character values

Table 9.1 continued

Option	Description
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
<i>statistic-keywords</i>	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 101.

ALPHA=*value*

specifies the confidence level to compute the confidence limits for the mean. The percentage for the confidence limits is $100(1 - \text{value})$. 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 103.

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 105.

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 100.

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 Programmers Guide: Essentials* 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 104.

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 104 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 111.

VARDEF=divisor

specifies the divisor to use in the calculation of the variance and standard deviation. [Table 9.2](#) shows the possible values for *divisor* and their associated formulas.

Table 9.2 Values for VARDEF= Option

<i>divisor</i>	Description	Formula for Divisor
DF	Degrees of freedom	$n - 1$
N	Number of observations	n
WDF	Sum of weights minus one	$(\sum_i w_i) - 1$
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 107.

Statistic Keywords

Optional *statistic-keywords* specify which statistics to compute and the order in which to display them in the output. [Table 9.3](#) 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 107.

Table 9.3 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 107.

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 20, *SAS/STAT User’s Guide*).

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 101.

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 102.

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 Programmers Guide: Essentials* 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 9.4](#) lists the *statistic-keywords* that are available in the OUTPUT statement.

Table 9.4 Statistics Keywords in the OUTPUT Statement

Descriptive Statistic Keywords	
CSS	RANGE
CV	SKEWNESS SKEW
KURTOSIS KURT	STDDEV STD
LCLM	STDERR
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 107.

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 computing and requests detailed results about the performance characteristics of the HPSUMMARY procedure.

The NTHREADS= option influences, but does not explicitly specify, how many threads the HPSUMMARY procedure uses. Setting the NTHREADS= option to a value greater than the CPU count or processor core count is not likely to improve overall performance.

The PERFORMANCE statement is documented further in the section “[PERFORMANCE Statement](#)” on page 5 of Chapter 2, “[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 9.5 illustrate grouping syntax:

Table 9.5 Examples of Grouping Syntax

Request	Equivalent To
<code>types A*(B C);</code>	<code>types A*B A*C;</code>
<code>types (A B)*(C D);</code>	<code>types A*C A*D B*C B*D;</code>
<code>types (A B C)*D;</code>	<code>types A*D B*D C*D;</code>

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. 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 105.

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 9.6 describes how PROC HPSUMMARY treats various values of the *weight-variable*.

Table 9.6 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 9.7](#) describes how PROC HPSUMMARY treats various values of *weight-variable*.

Table 9.7 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 HPSUMMARY 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 107 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) are 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

$$Nc_1(Lc_1 + K) + Nc_2(Lc_2 + K) + \cdots + Nc_n(Lc_n + K)$$

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 97. 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

$$W * Nc_1 * Nc_2 * \cdots * Nc_n$$

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 $Nc_1, \dots, Nc_n$ 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 107 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

$$\begin{aligned} \bar{x} + t_{(1-\alpha; n-1)} \frac{s}{\sqrt{n}} & \quad (\text{upper}) \\ \bar{x} - t_{(1-\alpha; n-1)} \frac{s}{\sqrt{n}} & \quad (\text{lower}) \end{aligned}$$

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 i th 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

$$t_w = \frac{\bar{y}_w - \mu_0}{s_w / \sqrt{\sum_{i=1}^n w_i}}$$

where $\bar{y}_w$ is the weighted mean, s_w is the weighted standard deviation, and w_i is the weight for the i th 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 111. 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 HPSUMMARY 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 HPSUMMARY` 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 9.8 shows the values of `_TYPE_` and the number of observations in the data set when you specify one, two, and three classification variables.

Table 9.8 The Effect of Classification Variables on the OUTPUT Data Set

CLASS Variables			_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
C	B	A					
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

$$\sum w_i x_i / \sum w_i$$

 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	$(\sum_i w_i) - 1$

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

$$c_{4n} \sum z_i^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

where c_{4n} 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 $w_i^* = zw_i$, $z > 0$. 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

$$c_{3n} \sum z_i^3$$

where c_{3n} 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 $w_i^* = zw_i$, $z > 0$. 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

$$\sum w_i x_i$$

SUMWGTis the sum of the weights, W , computed as

$$\sum w_i$$

USS

is the uncorrected sum of squares, computed as

$$\sum w_i x_i^2$$

VARis 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 1st percentile, P5 is the 5th 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 $x_1, x_2, \dots, x_n$ 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 t th 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

$$\begin{aligned} np &= j + g && \text{when QNTLDEF=1, 2, 3, or 5} \\ (n + 1)p &= j + g && \text{when QNTLDEF=4} \end{aligned}$$

Here, QNTLDEF= specifies the method that the procedure uses to compute the t th percentile, as shown in Table 9.10.

When you use the WEIGHT statement, the t th 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 9.10 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

$$\frac{\bar{x} - \mu_0}{s / \sqrt{\sum w_i}}$$

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

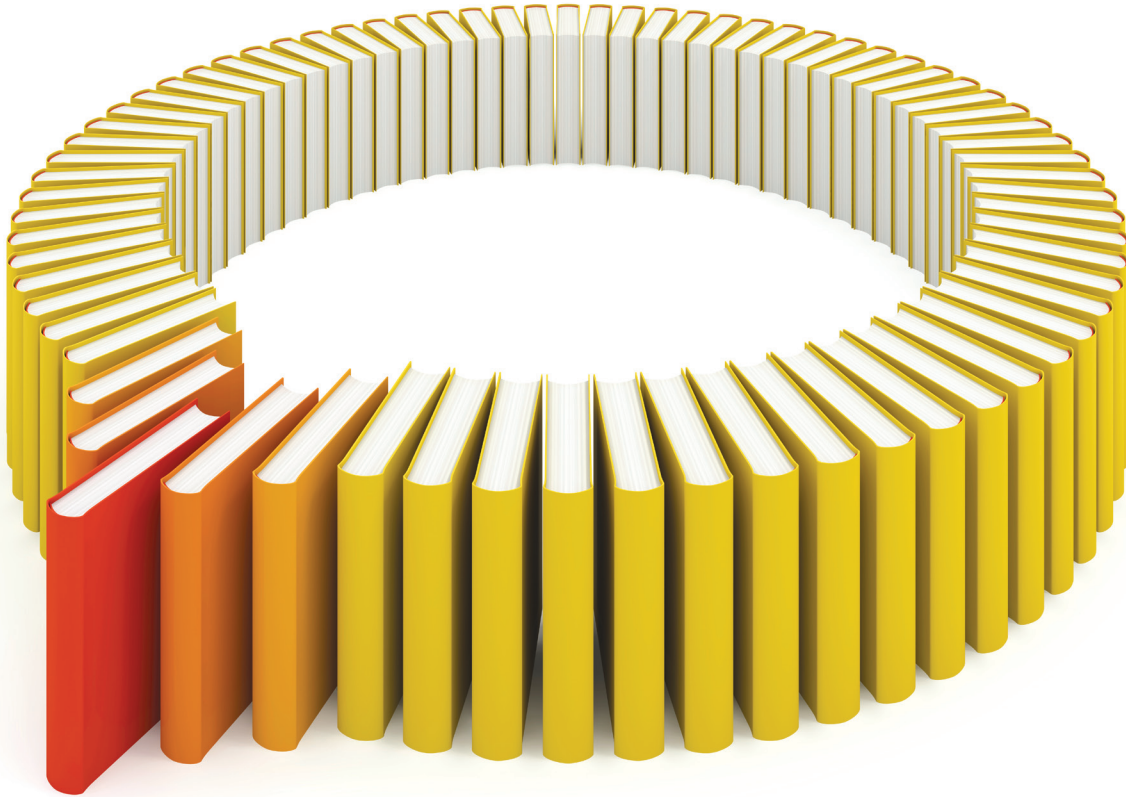
The following are the minimal data requirements to compute unweighted statistics and do not describe recommended sample sizes. Statistics are reported as missing if **VARDEF=DF** (the default) and the following requirements are not met:

- 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.



Gain Greater Insight into Your SAS[®] Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.



support.sas.com/bookstore
for additional books and resources.

