Paper 3-2018

Essential Programming Techniques Every SAS® User Should Learn

Kirk Paul Lafler, Software Intelligence Corporation

Abstract

SAS® software boasts countless functions, algorithms, procedures, options, methods, code constructs, and other features to help users automate and deploy solutions for specific tasks and problems, as well as to access, transform, analyze, and manage data. This paper identifies and shares essential programming techniques that a pragmatic user and programmer should learn. Topics include determining the number of by-group levels that exist within classification variables; data manipulation with the family of CAT functions; merging or joining multiple tables of data; performing table lookup operations with user-defined formats; creating single-value and value-list macro variables with PROC SQL; examining and processing the contents of value-list macro variables; determining the FIRST., LAST. and Between by-group rows; processing repetitive data with arrays; and using metadata to better understand the contents of SAS datasets.

Introduction

As SAS users around the world celebrate a milestone of more than 40-years using SAS software, users should learn as many essential programming techniques as possible to enhance their careers well into the 21st century and beyond. Whether you're a beginner who's just started out learning how to use SAS, an intermediate or an advanced user who has developed code, programs and/or applications using SAS software, you should do everything possible to expand your skillset. The good news is that there is no shortage of ways to learn SAS including the many SAS communities, such as communities.sas.com, blogs.sas.com, www.lexjansen.com, and many others. From new techniques to new technologies, most will help you immensely as you continue to pursue your learning objectives.

Essential Programming Techniques Every SAS User Should Learn

The most common response to the question, "What essential programming techniques should SAS users learn?" varies depending on who you ask. The fact is if you were to ask ten different SAS users what essential programming techniques a SAS user should learn you'd most likely receive a variety of responses. For some, essential programming techniques include arrays, faster programming constructs and table lookups. For others, essential programming techniques include modernizing outdated, statements, functions, options, coding constructs, algorithms and other techniques with newer, faster and more scalable programming techniques. So, what essential SAS programming techniques should be learned? In an attempt to shed some light on this very important question, I have shared a few topics, below.

Conditional Logic Scenarios

A powerful and necessary programming technique in the SAS® software is its ability to perform different actions depending on whether a programmer-specified condition evaluates to true or false. The method for accomplishing this is to use one or more conditional statements, expressions, and constructs to build a level of intelligence in a program or application. Conditional logic scenarios in the DATA step are frequently implemented using IF-THEN / ELSE and SELECT statements. The SQL procedure also supports logic scenarios and is implemented with a coding technique known as a CASE expression.

Conditional Logic with IF-THEN / ELSE

The IF-THEN / ELSE construct in the DATA step enables a sequence of conditions to be assigned that when executed proceeds through the sequence of logic conditions until a match in an expression is found or until all conditions are exhausted. The example shows a character variable Movie_Length being assigned a value of either "Shorter Length", "Average Length", or "Longer Length" based on the mutually exclusive conditions specified in the IF-THEN and ELSE conditions. Although not required, an ELSE condition serves as an effective technique for continuing processing to the next specified condition when a match is not found. An ELSE condition can also be useful as a "catch-all" to prevent a missing value from being assigned.

IF-THEN / ELSE Code:

```
LIBNAME MYDATA "E:/WORKSHOPS/WORKSHOP DATA" ;
DATA IF_THEN_EXAMPLE ;
  ATTRIB Movie_Length LENGTH=$14 LABEL='Movie Length' ;
  SET MYDATA.MOVIES ;
  IF LENGTH < 120 THEN Movie_Length = 'Shorter Length' ;
  ELSE IF LENGTH > 160 THEN Movie_Length = 'Longer Length' ;
  ELSE Movie_Length = 'Average Length' ;
RUN ;
PROC PRINT DATA=IF_THEN_EXAMPLE NOOBS ;
  VAR TITLE LENGTH Movie_Length ;
RUN ;
```

IF-THEN / ELSE Results:

Title	Length	Movie_Length
Brave Heart	177	Longer Length
Casablanca	103	Shorter Length
Christmas Vacation	97	Shorter Length
Coming to America	116	Shorter Length
Dracula	130	Average Length
Dressed to Kill	105	Shorter Length
Forrest Gump	142	Average Length
Ghost	127	Average Length
Jaws	125	Average Length
Jurassic Park	127	Average Length
Lethal Weapon	110	Shorter Length
Michael	106	Shorter Length
National Lampoon's Vacation	98	Shorter Length
Poltergeist	115	Shorter Length
Rocky	120	Average Length
Scarface	170	Longer Length
Silence of the Lambs	118	Shorter Length
Star Wars	124	Average Length
The Hunt for Red October	135	Average Length
The Terminator	108	Shorter Length
The Wizard of Oz	101	Shorter Length
Titanic	194	Longer Length

Conditional Logic with SELECT-WHEN / OTHERWISE

Another form of conditional logic available to users is a **SELECT** statement. Its purpose is to enable a sequence of logic conditions to be constructed in a DATA step by specifying one or more **WHEN** conditions and an optional **OTHERWISE** condition. When executed, processing continues through each WHEN condition until a match is found that satisfies the specified expression. Typically one or more WHEN conditions are specified in descending frequency order representing a series of conditions. The next example shows a value based on the mutually exclusive conditions specified in the sequence of logic conditions of "Shorter Length", "Average Length", or "Longer Length" being assigned to the character variable Movie_Length. Although not required, the OTHERWISE condition can be useful in the assignment of a specific value or as a "catch-all" to prevent a missing value from being assigned.

SELECT-WHEN / OTHERWISE Code:

```
LIBNAME MYDATA "E:/WORKSHOPS/WORKSHOP DATA" ;
DATA SELECT_EXAMPLE ;
  SET MYDATA.MOVIES ;
  SELECT ;
    WHEN (LENGTH < 120) Movie_Length = 'Shorter Length' ;
    WHEN (LENGTH > 160) Movie_Length = 'Longer Length' ;
    OTHERWISE Movie_Length = 'Average Length' ;
  END ;
RUN ;

PROC PRINT DATA=SELECT_EXAMPLE NOOBS ;
  VAR TITLE LENGTH Movie_Length ;
RUN ;
```

SELECT-WHEN / OTHERWISE Results:

Title	Length	Movie_Length
Brave Heart	177	Longer Length
Casablanca	103	Shorter Length
Christmas Vacation	97	Shorter Length
Coming to America	116	Shorter Length
Dracula	130	Average Length
Dressed to Kill	105	Shorter Length
Forrest Gump	142	Average Length
Ghost	127	Average Length
Jaws	125	Average Length
Jurassic Park	127	Average Length
Lethal Weapon	110	Shorter Length
Michael	106	Shorter Length
National Lampoon's Vacation	98	Shorter Length
Poltergeist	115	Shorter Length
Rocky	120	Average Length
Scarface	170	Longer Length
Silence of the Lambs	118	Shorter Length
Star Wars	124	Average Length
The Hunt for Red October	135	Average Length
The Terminator	108	Shorter Length
The Wizard of Oz	101	Shorter Length
Titanic	194	Longer Length

Conditional Logic with CASE Expressions

Another form of conditional logic available to users is a case expression. Its purpose is to provide a way of conditionally selecting result values from each row in a table (or view). Similar to an IF-THEN/ELSE or SELECT construct in the DATA step, a case expression can only be specified in the SQL procedure. It supports a WHEN-THEN clause to conditionally process some but not all the rows in a table. An optional ELSE expression can be specified to handle an alternative action should none of the expression(s) identified in the WHEN condition(s) not be satisfied. A case expression must be a valid SQL expression and conform to syntax rules similar to DATA step SELECT-WHEN statements. Even though this topic is best explained by example, a quick look at the syntax follows.

```
CASE <column-name>
   WHEN when-condition THEN result-expression
  <WHEN when-condition THEN result-expression> …

  <ELSE result-expression>
END
```

A column-name can optionally be specified as part of the CASE-expression. If present, it is automatically made available to each when-condition, and is classified as a simple case expression. When it is not specified, the column-name must be coded in each when-condition, and is classified as a searched case expression. If a when-condition is satisfied by a row in a table (or view), then it is considered "true" and the result-expression following the THEN keyword is processed. The remaining WHEN conditions in the case expression are skipped. If a when-condition is "false", the next when-condition is evaluated. SQL evaluates each when-condition until a "true" condition is found or in the event all when-conditions are "false", it then executes the ELSE expression and assigns its value to the CASE expression's result. A missing value is assigned to a case expression when an ELSE expression is not specified and each when-condition is "false".

In the next example, a **searched case expression** is illustrated. A searched case expression in the SQL procedure provides users with the capability to perform more complex comparisons. Although the number of keystrokes can be more than with a simple case expression, the searched case expression offers the greatest flexibility and is the primary form used by SQL'ers. The noticeable absence of a column name as part of the case expression permits any number of columns to be specified from the underlying table(s) in the WHEN-THEN/ELSE logic scenarios.

The next example shows a searched case expression being used to assign the character variable Movie_Length with the AS keyword. A value of "Shorter Length" for movie lengths less than 120 minutes, "Longer Length" for movie lengths greater than 160 minutes, or "Average Length" for all other movie lengths is assigned to the newly created column. Although not required, an ELSE condition can be useful in the assignment of a specific value or as a "catch-all" to prevent a missing value from being assigned.

Searched CASE Expression Code:

```
LIBNAME MYDATA "E:/WORKSHOPS/WORKSHOP DATA" ;
PROC SQL;
  SELECT TITLE,
         LENGTH,
         CASE
           WHEN LENGTH < 120 THEN 'Shorter Length'
           WHEN LENGTH > 160 THEN 'Longer Length'
           ELSE 'Average Length'
         END AS Movie_Length
    FROM MYDATA.MOVIES ;
QUIT ;
```

Searched CASE Expression Results:

Title	Length	Movie_Length
Brave Heart	177	Longer Length
Casablanca	103	Shorter Length
Christmas Vacation	97	Shorter Length
Coming to America	116	Shorter Length
Dracula	130	Average Length
Dressed to Kill	105	Shorter Length
Forrest Gump	142	Average Length
Ghost	127	Average Length
Jaws	125	Average Length
Jurassic Park	127	Average Length
Lethal Weapon	110	Shorter Length
Michael	106	Shorter Length
National Lampoon's Vacation	98	Shorter Length
Poltergeist	115	Shorter Length
Rocky	120	Average Length
Scarface	170	Longer Length
Silence of the Lambs	118	Shorter Length
Star Wars	124	Average Length
The Hunt for Red October	135	Average Length
The Terminator	108	Shorter Length
The Wizard of Oz	101	Shorter Length
Titanic	194	Longer Length

As previously mentioned, searched case expressions provide users with the capability to perform more complex logic comparisons. Combined with logical and comparison operators, searched case expressions along with their WHERE clause counterparts, provide the capabilities to construct complex logic scenarios. In the next example a listing of “Action” and “Comedy” movies are displayed. Using a searched case expression, a value of “Shorter Length” for movie lengths less than 120 minutes, “Longer Length” for movie lengths greater than 160 minutes, or “Average Length” for all other movie lengths is assigned to the newly created column. A column heading of Movie_Type is assigned to the new column with the AS keyword.

Searched CASE Expression Code:

```
LIBNAME MYDATA “E:/WORKSHOPS/WORKSHOP DATA” ;
PROC SQL;
  SELECT TITLE, RATING, LENGTH, CATEGORY,
    CASE
      WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND LENGTH < 120 THEN 'Action Short'
      WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND LENGTH > 160 THEN 'Action Long'
      WHEN UPCASE(CATEGORY) CONTAINS 'ACTION' AND
           LENGTH BETWEEN 120 AND 160 THEN 'Action Medium'
      WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND LENGTH < 120 THEN 'Comedy Short'
      WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND LENGTH > 160 THEN 'Comedy Long'
      WHEN UPCASE(CATEGORY) CONTAINS 'COMEDY' AND
           LENGTH BETWEEN 120 AND 160 THEN 'Comedy Medium'
      ELSE 'Not Interested'
    END AS MOVIE_TYPE
    FROM MYDATA.MOVIES
      WHERE UPCASE(CATEGORY) CONTAINS 'ACTION' OR 'COMEDY';
    QUIT;
```

Searched CASE Expression Results:

Title	Rating	Length	Category	MOVIE_TYPE
Brave Heart	R	177	Action Adventure	Action Long
Casablanca	PG	103	Drama	Not Interested
Christmas Vacation	PG-13	97	Comedy	Comedy Short
Coming to America	R	116	Comedy	Comedy Short
Dracula	R	130	Horror	Not Interested
Dressed to Kill	R	105	Drama Mysteries	Not Interested
Forrest Gump	PG-13	142	Drama	Not Interested
Ghost	PG-13	127	Drama Romance	Not Interested
Jaws	PG	125	Action Adventure	Action Medium
Jurassic Park	PG-13	127	Action	Action Medium
Lethal Weapon	R	110	Action Cops & Robber	Action Short
Michael	PG-13	106	Drama	Not Interested
National Lampoon's Vacation	PG-13	98	Comedy	Comedy Short
Poltergeist	PG	115	Horror	Not Interested
Rocky	PG	120	Action Adventure	Action Medium
Scarface	R	170	Action Cops & Robber	Action Long
Silence of the Lambs	R	118	Drama Suspense	Not Interested
Star Wars	PG	124	Action Sci-Fi	Action Medium
The Hunt for Red October	PG	135	Action Adventure	Action Medium
The Terminator	R	108	Action Sci-Fi	Action Short
The Wizard of Oz	G	101	Adventure	Not Interested
Titanic	PG-13	194	Drama Romance	Not Interested

Subsetting with WHERE Expressions in a PROCedure

Gupta (2006) describes using a subsetting-IF versus a WHERE-statement or WHERE= data set option to subset observations. To avoid using a subsetting-IF statement in a DATA step, SAS users may be able to specify a WHERE= data set option for subsetting purposes directly in a procedure. This approach prevents the creation of a data set and, as a result, is more likely to scale better by reducing CPU and I/O resources. Gupta emphasizes an important detail that all SAS users should know when specifying a WHERE condition in a procedure, "Multiple WHERE conditions within SAS procedures are not cumulative as they are in a DATA step meaning the most recent WHERE condition replaces any, and all, previously specified WHERE condition(s)."

PROC PRINT with WHERE Expression Code:

```
/* WHERE Statement to Subset Observations */
proc print data=sashelp.cars noobs ;
  where type="SUV" or type="Wagon" ;
run ;

            < or >

/* WHERE= Data Set Option to Subset Observations */
proc print data=sashelp.cars(where=(type="SUV" or type="Wagon")) noobs ;
run ;
```

Using the IN Operator for Comparisons

Legacy SAS applications and program code often use one, or more, OR comparison operators to handle logic scenarios. Although syntactically correct, a series of individual comparisons separated by an OR comparison operator is generally less efficient than using an IN operator. The reason is due to the way an IN operator operates. When an IN operator is specified, SAS stops making comparisons as soon as it finds a match. This is not the case with an OR operator. In the next example, a number of individual comparisons are specified using an OR operator.

OR Comparison Operator Code:

```
PROC SQL ;
  SELECT Origin, Type, MSRP
   FROM SASHELP.Cars
    WHERE Type = "SUV"
       OR Type = "Truck"
       OR Type = "Wagon"
     ORDER BY MSRP ;
QUIT ;
```

In the next example, an IN operator is specified to help modernize the process of handling a number of individual comparisons. The IN operator provides a convenient, and concise, way to specify scenarios with many OR comparisons. *A similar example using DS2 SQLSTMT package approach can be found here.*

IN Operator Code:

```
PROC SQL ;
  SELECT Origin, Type, MSRP
   FROM SASHELP.Cars
    WHERE Type IN ( "SUV","Truck","Wagon" )
     ORDER BY MSRP ;
QUIT ;
```

Concatenating Strings and Variables with CAT Functions

SAS functions serve an essential role in the Base SAS software. Representing a variety of built-in and callable routines, functions serve as the “work horses” in the SAS software providing users with “ready-to-use” tools designed to ease the burden of writing and testing often lengthy and complex code for a variety of programming tasks. The advantage of using SAS functions is evident by their relative ease of use, and their ability to provide a more efficient, robust and scalable approach to simplifying a process or programming task. In this example, we show how the TRIM and LEFT functions along with the concatenate operator to concatenate strings and variables together can be replaced with the CAT functions.

CAT Function Code:

```
data _null_ ;
  length NUM 3. A B C D E $ 8 BLANK $ 1 ;
  A = 'The' ;
  NUM = 5 ;
  B = ' Cats' ;
  C = 'in' ;
  D = ' the' ;
  E = 'Hat' ;
  BLANK = ' ' ;
```

```
 *Old concatenation approach with TRIM and LEFT functions and concatenation
  operator ;
  OLD=trim(left(A)) || BLANK || trim(left(NUM)) || BLANK || trim(left(B)) ||
      BLANK || trim(left(C)) || BLANK || trim(left(D)) || BLANK || trim(left(E)) ;

  * Using the CAT functions to concatenate character and numeric values together ;
❶ CAT  = cat (A, NUM, B, C, D, E) ;
❷ CATQ = catq(BLANK, A, NUM, B, C, D, E) ;
❸ CATS = cats(A, NUM, B, C, D, E) ;
❹ CATT = catt(A, NUM, B, C, D, E) ;
❺ CATX = catx(BLANK, A, NUM, B, C, D, E) ;
   put OLD= / STRIP= / CAT= / CATQ= / CATS= / CATT= / CATX= / ;
run ;
```

CAT Function Results:

```
OLD=The 5 Cats in the Hat
CAT=The     5 Cats   in       the    Hat
CATQ="The     " 5 " Cats   " "in      " " the    " "Hat     "
CATS=The5CatsintheHat
CATT=The5 Catsin theHat
CATX=The 5 Cats in the Hat
```

Analysis:

In the example, above, a single numeric variable, NUM, and six character variables: A, B, C, D, E, and BLANK are defined with their respective values as: NUM=5, A='The', B=' Cats', C='in', D=' the', E='Hat' and BLANK=' '. The oldest way of concatenating two or more strings or variables together is then specified, using the TRIM and LEFT functions with the concatenation operator "||" in an assignment statement. As an alternative, a newer and more robust concatenation approach is specified using the CAT family of functions: CAT, CATQ, CATS, CATT, and CATX.

❶ **CAT**, the simplest of concatenation functions, joins two or more strings and/or variables together, end-to-end producing the same results as with the concatenation (double bar) operator.

❷ **CATQ** is similar to the default features of the CATX function, but the CATQ function adds quotation marks to any concatenated string or variable.

❸ **CATS** removes leading and trailing blanks and concatenates two or more strings and/or variables together.

❹ **CATT** removes trailing blanks and concatenates two or more strings and/or variables together.

❺ **CATX**, perhaps the most robust CAT function, removes leading and trailing blanks and concatenates two or more strings and/or variables together with a user-specified delimiter between each.

Concatenating (or Appending) Data Sets

Concatenating data sets is the process of combining two, or more, data sets, one after the other, with the purpose of creating a single data set. The number of observations in the new data set is the sum total of observations in all the original input data sets. The order of observations in the concatenated data set is arranged sequentially with the observations from the first data set, followed by the observations from the second data set, and so on. The concatenated data set contains the same variables as the input data sets. Should an input data set contain different variables from the other input data sets, the concatenated data set will have missing values assigned to the variables from the other input data sets.

Concatenating with the DATA-SET-RUN Construct

SAS provides users with a few ways to concatenate data sets. In the first example, below, an old-style DATA-SET construct is specified to concatenate the two data sets, RUGs_2015 and RUGs_2016. Although syntactically correct, this approach does not scale well because it forces SAS to incur heavy I/O (input/output) because the observations in each input data set must be read and written to the concatenated data set.

DATA–SET-RUN Code:

```
data Concatenated_Results ;
  set RUGs_2015
      RUGs_2016 ;
run ;
```

DATA–SET-RUN Results:

RUG	Number_Papers	Year
MWSUG	96	2015
SCSUG	29	2015
SESUG	148	2015
WUSS	102	2015
MWSUG	124	2016
SCSUG	62	2016
SESUG	148	2016
WUSS	112	2016

Concatenating with a PROC SQL Outer Union CORR

A second approach uses PROC SQL to concatenate data sets. In this next example, an OUTER UNION CORR set operator is specified, and SQL reads and processes the tables in each query producing a new concatenated table of results.

PROC SQL Code:

```
proc sql ;
 create table Concatenated_Results as
  select * from RUGs_2015
 outer union corr
  select * from RUGs_2016 ;
  select * from Concatenated_Results ;
quit ;
```

PROC SQL Results:

RUG	Number_Papers	Year
MWSUG	96	2015
SCSUG	29	2015
SESUG	148	2015
WUSS	102	2015
MWSUG	124	2016
SCSUG	62	2016
SESUG	148	2016
WUSS	112	2016

Concatenating with PROC APPEND (or PROC DATASETS – APPEND Statement)

A third, and more efficient, concatenation approach is available to SAS users. Using PROC APPEND (or the APPEND statement in PROC DATASETS), an input data set can be appended to another data set. The advantage of using this approach is reduced I/O, since SAS does not have to read the observations in the base data set. Appending this way offers a way to scale an application. As the number of observations in the base data set grows, the advantage of using this approach can become huge. In the next example, two PROC APPENDs are specified to concatenate the observations in the RUGs_2015 and RUGs_2016 data sets.

PROC APPEND Code:

```
proc append base=Concatenated_Results
            Data=RUGs_2015 ;
run ;
proc append base=Concatenated_Results
            Data=RUGs_2016 ;
run ;
```

PROC APPEND Results:

RUG	Number_Papers	Year
MWSUG	96	2015
SCSUG	29	2015
SESUG	148	2015
WUSS	102	2015
MWSUG	124	2016
SCSUG	62	2016
SESUG	148	2016
WUSS	112	2016

Processing Multiple TABLE Statements with PROC FREQ

Benjamin (2012) describes a common problem programmers have when using PROC FREQ to produce multiple table results. Programmers will often code two, or more, individual PROC FREQ and TABLE statements even for the same input data set. Although the PROC FREQ code, illustrated below, is syntactically correct, invoking PROC FREQ multiple times in this way can result in an increase in the amount of time for processing the request.

PROC FREQ Code:

```
proc freq data=sashelp.cars ;
  table Origin / list out=work.Origin_Freq1 ;
run ;
proc freq data=sashelp.cars ;
  table Origin * Type / list out=work.Origin_Freq2 ;
run ;
proc freq data=sashelp.cars ;
  table Origin * Type * Cylinders / list out=work.Origin_Freq3 ;
run ;
```

To optimize the code, programmers can force a single pass over the input data set and as a result reduce the amount of processing time needed to produce the resulting data sets, as follows.

Optimized PROC FREQ Code:

```
proc freq data=sashelp.cars ;
  table Origin / list out=work.Origin_Freq1 ;
  table Origin * Type / list out=work.Origin_Freq2 ;
  table Origin * Type * Cylinders / list out=work.Origin_Freq3 ;
run ;
```

List of Procedures Supporting a CLASS Statement

Procedures are classified as the "workhorses" in the SAS System. The CLASS statement specifies one, or more, character or numeric variables used to group data into classification levels. A virtue of using a CLASS statement is that a SORT procedure is not required to arrange and group the data, because the stats and other information is collected in memory and reported at the end of the procedure. A partial list of SAS procedures, below, supports the use of a CLASS statement.

SAS Procedures Supporting a CLASS Statement			
PROC ANOVA	PROC MEANS	PROC REPORT	PROC TTEST
PROC DISCRIM	PROC MIXED	PROC SUMMARY	PROC UNIVARIATE
PROC GENMOD	PROC NESTED	PROC SURVEYMEANS	
PROC GLM	PROC PHREG	PROC TABULATE	
PROC LOGISTIC	PROC REG	PROC TIMEPLOT	

Producing Page Numbers with ODS RTF Pagination Functions

Page numbering is the process of applying a sequence of numbers, Roman numerals, or letters on reports, spreadsheets, documents, books or other multi-page files. Legacy applications and program code frequently use counters or code routines to generate and display page numbers. Simple page numbering routines may resemble something similar to the following code.

DATA Step Code:

```
FILENAME REPORT DISK 'c:\DATA_NULL_Report.LST' ;
DATA _NULL_ ;
  SET SASHELP.CARS END=EOF ;
  FILE REPORT HEADER=H1 ; /* Execute Page_Header Routine */
  PUT @1 Origin  $6.
     @10 Make    $13.
     @25 MSRP    DOLLAR12. ;
RETURN ;

H1: ; /* Page Header */
  Page_CTR + 1 ;
  PUT @15 DATA _NULL_ Detail Report
   // @22 'Page Number ' Page_CTR  ;
RETURN ;
RUN ;
```

Page numbers can be produced and displayed in RTF output by specifying an escape character with an ODS RTF statement, any of the following functions, and an ODS RTF CLOSE ; statement:

- ✓ **{thispage}**
- ✓ **{lastpage}**
- ✓ **{pageof}**

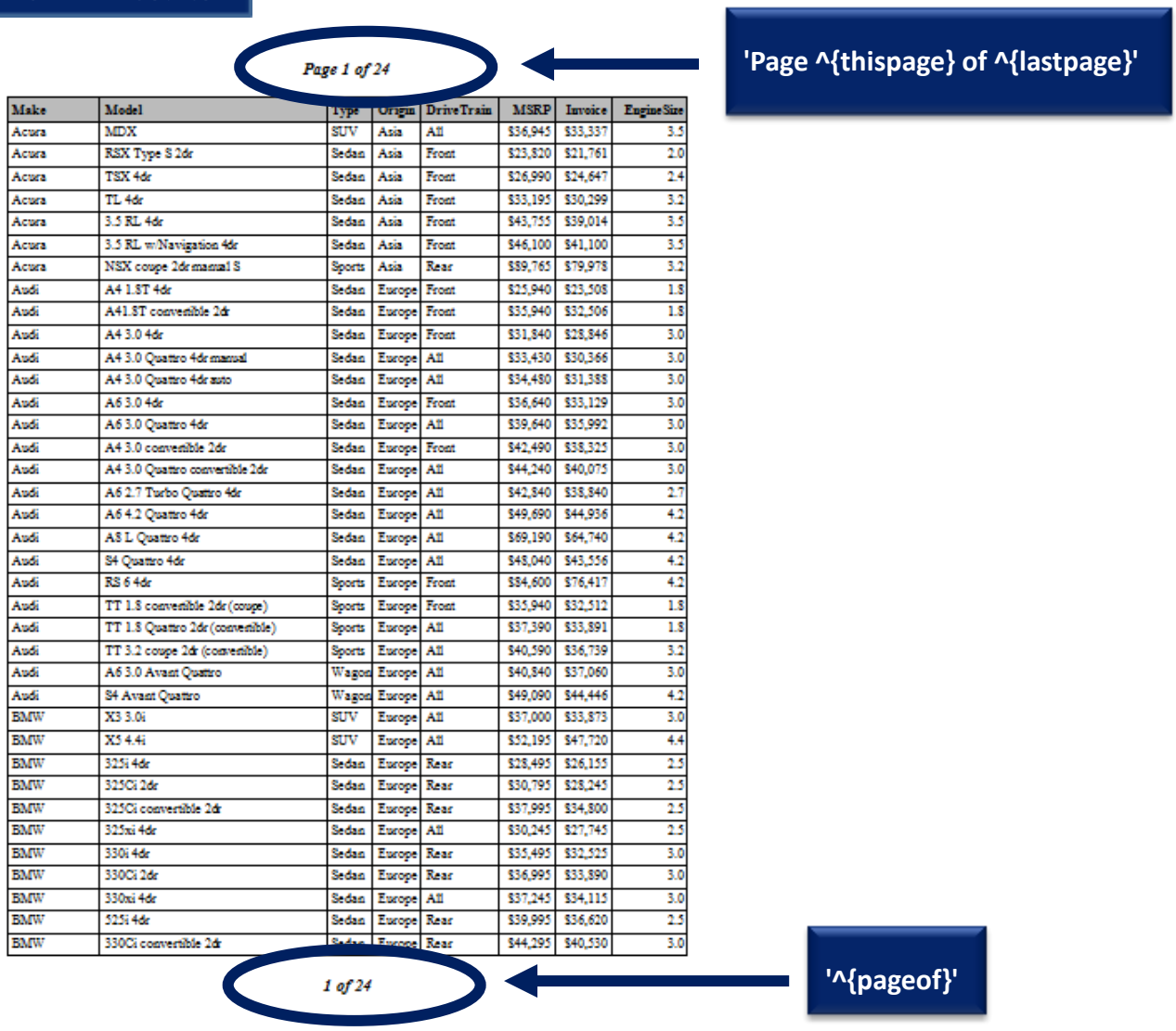
Page Counters with ODS RTF Functions

Output Delivery System (ODS) provides powerful features that users can use when producing output. In the next example, an escape character is specified with the ODS RTF destination, where the functions: {thispage}, {lastpage}, and {pageof} are specified in the title and footnote statements to produce the page numbers and the total number of pages in the report.

ODS RTF Code:

```
ods escapechar='^' ;
ods RTF file='c:\Print-Report.rtf' ;
 proc print data=sashelp.cars noobs ;
   title 'Page ^{thispage} of ^{lastpage}' ;
   footnote '^{pageof}' ;
 run ;
ods RTF close ;
```

ODS RTF Results:



Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize
Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5
Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2.0
Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4
Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2
Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5
Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5
Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2
Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8
Audi	A41.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8
Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3.0
Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3.0
Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3.0
Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3.0
Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3.0
Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	$42,490	$38,325	3.0
Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	$44,240	$40,075	3.0
Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	$42,840	$38,840	2.7
Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	$49,690	$44,936	4.2
Audi	A8 L Quattro 4dr	Sedan	Europe	All	$69,190	$64,740	4.2
Audi	S4 Quattro 4dr	Sedan	Europe	All	$48,040	$43,556	4.2
Audi	RS 6 4dr	Sports	Europe	Front	$84,600	$76,417	4.2
Audi	TT 1.8 convertible 2dr (coupe)	Sports	Europe	Front	$35,940	$32,512	1.8
Audi	TT 1.8 Quattro 2dr (convertible)	Sports	Europe	All	$37,390	$33,891	1.8
Audi	TT 3.2 coupe 2dr (convertible)	Sports	Europe	All	$40,590	$36,739	3.2
Audi	A6 3.0 Avant Quattro	Wagon	Europe	All	$40,840	$37,060	3.0
Audi	S4 Avant Quattro	Wagon	Europe	All	$49,090	$44,446	4.2
BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3.0
BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4
BMW	325i 4dr	Sedan	Europe	Rear	$28,495	$26,155	2.5
BMW	325Ci 2dr	Sedan	Europe	Rear	$30,795	$28,245	2.5
BMW	325Ci convertible 2dr	Sedan	Europe	Rear	$37,995	$34,800	2.5
BMW	325xi 4dr	Sedan	Europe	All	$30,245	$27,745	2.5
BMW	330i 4dr	Sedan	Europe	Rear	$35,495	$32,525	3.0
BMW	330Ci 2dr	Sedan	Europe	Rear	$36,995	$33,890	3.0
BMW	330xi 4dr	Sedan	Europe	All	$37,245	$34,115	3.0
BMW	525i 4dr	Sedan	Europe	Rear	$39,995	$36,620	2.5
BMW	330Ci convertible 2dr	Sedan	Europe	Rear	$44,295	$40,530	3.0

Automating the Process of Creating Multiple HTML Files

The Web offers incredible potential that impacts all corners of society. With its increasing popularity as a communications medium, Web publishers have arguably established the Web as the greatest medium ever created. Businesses, government agencies, professional associations, schools, libraries, research agencies, and a potpourri of society's true believers have endorsed the Web as an efficient means of conveying their messages to the world.

The SAS software provides users with the capability to create results and deploy selected pieces of output as HTML output files. Using the Output Delivery System (ODS) HTML destination, output can be created that anyone can view using a web browser. Syntactically correct HTML code is automatically produced and made ready for deployment using one of the Internet browser software products (e.g., Internet Explorer, Google Chrome, Mozilla FireFox, Safari, etc.). As a result, the SAS System and the HTML destination create a type of "streaming" or continuous output by adding elevator bars (horizontal and/or vertical) for easy navigation.

In the following example, redundant code and hardcoding issues are avoided by using PROC SQL to determine the number of unique (or distinct) values of the Origin column exist and once known are assigned to single-value and value-list macro variables. With the unique values assigned to two macro variables, an iterative %DO statement is specified to control the propagation of one, or more, HTML files containing one-way frequency results. The results of the three distinct HTML files that were created are also displayed, below.

ODS HTML Code:

```
/* Output HTML Files Location */
filename odsout "E:\" ;

options symbolgen ;
%macro multfiles ;
  proc sql noprint ;
    select count(distinct origin)
      into :morigin_cnt  /* derive number of origins    */
        from sashelp.cars
          order by origin ;
    select distinct origin
      into :morigin_list separated by "~" /* derive unique origin values */
        from sashelp.cars
          order by origin ;
  quit ;

  %do i=1 %to &morigin_cnt ;
    ods html path=odsout (URL=NONE)
             file="%SCAN(&morigin_list,&i,~)_FrequencyReport (MultiHTMLFiles).html"
            style=styles.barrettsblue ;
    title "Cars with Origin in %SCAN(&morigin_list,&i,~)" ;
    proc freq data=sashelp.cars(where=(origin = "%SCAN(&morigin_list,&i,~)")) ;
      tables type ;
      format msrp dollar12.0 ;
    run ;
    quit ;
    title ;
    ods html close ;
  %end ;
  %put &morigin_list ;
%mend multfiles ;

%multfiles ;
```

ODS HTML Results:

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	1.90	3	1.90
SUV	25	15.82	28	17.72
Sedan	94	59.49	122	77.22
Sports	17	10.76	139	87.97
Truck	8	5.06	147	93.04
Wagon	11	6.96	158	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	10	8.13	10	8.13
Sedan	78	63.41	88	71.54
Sports	23	18.70	111	90.24
Wagon	12	9.76	123	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	25	17.01	25	17.01
Sedan	90	61.22	115	78.23
Sports	9	6.12	124	84.35
Truck	16	10.88	140	95.24
Wagon	7	4.76	147	100.00

Automating the Process of Creating Multiple Excel Files

Statistics show that the world's most used software application is Microsoft Excel®. Due to this dominance, SAS provides users with several ways to send results, tables, statistics, images and other output directly to an Excel spreadsheet. In the next example, redundant code and hardcoding issues are avoided by using PROC SQL to determine the number of unique (or distinct) values of the Origin column and, once known, are assigned to single-value and value-list macro variables. With the values assigned to the two macro variables, an iterative %DO statement is specified to control the propagation of Excel files containing one-way frequency results. The results of the three distinct Excel files that were created are also displayed, below.

ODS Excel Code:

```
%macro multExcelfiles ;
  proc sql noprint ;
    select count(distinct origin)
      into :morigin_cnt  /* derive number of origins    */
        from sashelp.cars
          order by origin ;
    select distinct origin
      into :morigin_list separated by "~" /* derive unique origin values */
        from sashelp.cars
          order by origin ;
  quit ;

  %do i=1 %to &morigin_cnt ;
    ods Excel file="e:/%SCAN(&morigin_list,&i,~)_FreqReport (MultiExcelFiles).xlsx"
             style=styles.barrettsblue ;
    title "Cars with Origin in %SCAN(&morigin_list,&i,~)" ;
    proc freq data=sashelp.cars(where=(origin = "%SCAN(&morigin_list,&i,~)")) ;
      tables type ;
      format msrp dollar12.0 ;
    run ;
    quit ;
    title ;
    ods Excel close ;
  %end ;
  %put &morigin_list ;
%mend multExcelfiles ;

%multExcelfiles ;
```

ODS Excel Results:

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Hybrid	3	1.90	3	1.90
SUV	25	15.82	28	17.72
Sedan	94	59.49	122	77.22
Sports	17	10.76	139	87.97
Truck	8	5.06	147	93.04
Wagon	11	6.96	158	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	10	8.13	10	8.13
Sedan	78	63.41	88	71.54
Sports	23	18.70	111	90.24
Wagon	12	9.76	123	100.00

Type	Frequency	Percent	Cumulative Frequency	Cumulative Percent
SUV	25	17.01	25	17.01
Sedan	90	61.22	115	78.23
Sports	9	6.12	124	84.35
Truck	16	10.88	140	95.24
Wagon	7	4.76	147	100.00

Discovering the Number of Occurrences of a Value in a Data Set

Discovering the number of occurrences of individual values in a data set is useful information, particularly when constructing data-driven approaches. SAS provides several ways to count and determine the number of occurrences of a value in a data set.

Discovering the Number of Occurrences of a Value in a DATA Step

One approach for discovering the number of occurrences of a variable's value(s) is to construct a DATA step counting routine. In the next example, individual counters for the number of females and males are created, and after the last observation is read and processed, the results for each counter is output to the Counts data set, and the results displayed with PROC PRINT.

DATA Step Code:

```
data Counts(drop=Sex) ;
  set sashelp.Heart(keep=Sex) end=EOF ;
  if Sex = "Female" then Number_Females + 1 ;
  else if Sex = "Male" then Number_Males + 1 ;
  if EOF then do ;
     Total = Number_Females + Number_Males ;
     format Number_Females Number_Males Total comma7. ;
     output ;
  end ;
run ;
proc print data=Counts noobs ;
run ;
```

DATA Step Results:

Number_Females	Number_Males	Total
2,873	2,336	5,209

Discovering the Number of Occurrences of a Value with the PROC FREQ NLEVELS Option

Another approach for counting the number of occurrences of a variable's value(s) is to specify the NLEVELS option in PROC FREQ. In this example, the variable SEX is kept and the NLEVELS option is specified for the SASHELP.Heart data set. The results show there are two levels for the variable, SEX, with 2,873 females and 2,336 males.

PROC FREQ Code:

```
proc freq data=sashelp.Heart(keep=sex) NLEVELS ;
run ;
```

PROC FREQ Results:

The FREQ Procedure

Number of Variable Levels	
Variable	Levels
Sex	2

Sex	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Female	2873	55.15	2873	55.15
Male	2336	44.85	5209	100.00

Discovering the Number of Occurrences of a Value with PROC SQL

Another approach for counting the number of occurrences of a variable's value is to use the SUM function with an equality expression in PROC SQL. PROC SQL's data access and query capabilities offer SAS users with a powerful approach to summing down rows and across columns. In this example, a SELECT query is specified with a SUM function for counting the number of "Females", "Males" and their combined totals that are found in the SASHELP.HEART data set. An optional FORMAT=COMMA7. parameter is also specified to make the results easier to read.

PROC SQL Code:

```
proc sql ;
  select SUM(sex="Female") AS Number_Females format=comma7.,
         SUM(sex="Male")   AS Number_Males   format=comma7.,
         SUM(sex IN ("Female","Male")) AS Total format=comma7.
    from sashelp.Heart ;
quit ;
```

PROC SQL Results:

Number_Females	Number_Males	Total
2,873	2,336	5,209

Using Metadata to Determine the Number of Observations in a Data Set

Metadata is everywhere and is defined as information that describes data. Other definitions include information about data, or information about the design and specification of objects and data structures. In its most basic form, metadata is found in the cataloging systems of every academic library, public library, school library, and special library in the world. The typical book, magazine, microfiche, digital file, image, or object's metadata is stored in cataloging systems. These cataloging systems are not composed of words, sentences, paragraphs, or chapters, but contain information about its author(s), title, subject, keyword(s), description, publisher, publication date, ISBN, format, resource identifier, copyright, and other information.

Older Methods of Determining the Number of Observations in a Data Set

Before the availability of metadata in the SAS System, users developed and included code routines that determined the number of observations in a data set. An often used DATA step approach, since the beginning of SAS-time, constructs a variable that counts the number of observations. Although syntactically correct, this approach does not "scale" well – due to the amount of I/O incurred and the sizes of data sets – when computing the counter. The the next example, a DATA step approach computes the total number of "Sedans" found in the SASHELP.CARS data set, and displays the results using PROC PRINT.

DATA Step Code:

```
data sedans_counter(keep=type obs_ctr)
     cars_sedans(drop=obs_ctr) ;
  set sashelp.cars(keep=origin type make MSRP) end=eof ;
  where upcase(type) = "SEDAN" ;
  obs_ctr + 1 ;
  output cars_sedans ;
  if eof then output sedans_counter ;
run ;

proc print data=sedans_counter noobs ;
run ;
```

DATA Step Results:

Type	obs_ctr
Sedan	262

Using DICTIONARY.TABLES Metadata to Determine the Number of Observations in a Data Set

The SAS System collects and populates valuable metadata about SAS libraries, data sets (tables), catalogs, indexes, macros, system options, titles, views and other useful information in a collection of read-only tables called Dictionary tables. Dictionary tables serve a special purpose for SAS users by providing system-related information about the current SAS session's SAS databases and applications. When a query processes a Dictionary table, SAS automatically launches a discovery process at runtime to collect information pertinent to that table. This information is made available any time after a SAS session is started.

When users need more information about SAS data sets the TABLES Dictionary table can be very helpful. The TABLES Dictionary table provides detailed information about the library names, the member (or data set) names, the date a data set was created and last modified, the number of observations in a data set, and much more. The next example illustrates a popular approach that accesses the metadata content from the DICTIONARY.TABLES table to determine the number of observations in any SAS data set.

PROC SQL Code:

```
title "Number of Rows in a Table" ;

proc sql ;
  select libname, memname, nobs format=comma10.
    from Dictionary.Tables
      where nobs NE . ;
quit ;

title ;
```

PROC SQL Results:

Number of Rows in a Table

Library Name	Member Name	Number of Physical Observations
WORK	COUNTS	1
MYDATA	ACTORS	13
MYDATA	ACTORS_WITH_MESSY_DATA	15
MYDATA	AE	13
MYDATA	DM	24
MYDATA	EX	146
MYDATA	MOVIES	22
MYDATA	MOVIES_WITH_MESSY_DATA	31
SASHELP	AACOMP	2,020
SASHELP	AARFM	195
SASHELP	ADSMSG	426
SASHELP	AFMSG	1,090
SASHELP	AIR	144
SASHELP	APPLIANC	156
SASHELP	ASSCMGR	402
SASHELP	BASEBALL	322
SASHELP	BEI	24,205
SASHELP	BMIMEN	3,264
SASHELP	BMT	137
SASHELP	BURROWS	24,591
SASHELP	BUY	11
SASHELP	BWEIGHT	50,000
SASHELP	CARS	428
SASHELP	CITIDAY	1,069
SASHELP	CITIMON	145
SASHELP	CITIQTR	48
SASHELP	CITIWK	319
SASHELP	CITIYR	10
SASHELP	CLASS	19
SASHELP	CLASSFIT	19
SASHELP	QUAKES	15,578
SASHELP	RENT	10
SASHELP	RETAIL	58
SASHELP	REVHUB2	72
SASHELP	ROCKPIT	6
SASHELP	SASMBC	0
SASHELP	SASMSG	794
SASHELP	SHOES	395
SASHELP	SLKWXL	1,703
SASHELP	SMEMSG	34
SASHELP	SPRINGS	1,587
SASHELP	STEEL	44
SASHELP	STOCKS	699
SASHELP	STTMSG	336
SASHELP	SVRTDIST	2,373
SASHELP	SYR1001	105
SASHELP	TABLE	8
SASHELP	THICK	75
SASHELP	TIMEDATA	40,330
SASHELP	TOURISM	29
SASHELP	USECON	252
SASHELP	VBPLAYRS	11
SASHELP	VERBMGR	19
SASHELP	VIDMSG	7
SASHELP	VOTE1980	3,107
SASHELP	WEBMSG	349
SASHELP	WORKERS	67
SASHELP	YR1001	126
SASHELP	YR111	126
SASHELP	ZHC	7,445
SASHELP	ZIPCODE	41,232
SASHELP	ZTC	18,161
SASHELP	_CMPIDX_	44

Using SASHELP.VTABLE Metadata to Determine the Number of Observations in a Data Set

SAS also provides users with metadata content in a number of SASHELP views. In this next example the number of observations in any SAS data set can be determined by accessing the NOBS metadata content in the SASHELP.VTABLE view. This metadata content can be displayed using any output-producing SAS procedure, as shown below.

PROC PRINT Code:

```
title "Number of Rows in a Table" ;

proc print data=sashelp.vtable noobs ;
  var libname memname nobs ;
  format nobs comma10. ;
  where nobs NE . ;
run ;

title ;
```

PROC PRINT Results:

Number of Rows in a Table

libname	memname	nobs
WORK	COUNTS	1
MYDATA	ACTORS	13
MYDATA	ACTORS_WITH_MESSY_DATA	15
MYDATA	AE	13
MYDATA	DM	24
MYDATA	EX	145
MYDATA	MOVIES	22
MYDATA	MOVIES_WITH_MESSY_DATA	31
SASHELP	AACOMP	2,020
SASHELP	AARFM	195
SASHELP	ADSMSG	426
SASHELP	AFMSG	1,090
SASHELP	AIR	144
SASHELP	APPLIANC	156
SASHELP	ASSCMGR	402
SASHELP	BASEBALL	322
SASHELP	BEI	24,205
SASHELP	BMIMEN	3,264
SASHELP	BMT	137
SASHELP	BURROWS	24,591
SASHELP	BUY	11
SASHELP	BWEIGHT	50,000
SASHELP	CARS	428
SASHELP	CITIDAY	1,069
SASHELP	CITIMON	145
SASHELP	CITIQTR	48
SASHELP	CITIWK	319
SASHELP	CITIYR	10
SASHELP	CLASS	19
SASHELP	CLASSFIT	19
SASHELP	QUAKES	15,578
SASHELP	RENT	10
SASHELP	RETAIL	58
SASHELP	REVHUB2	72
SASHELP	ROCKPIT	6
SASHELP	SASMBC	0
SASHELP	SASMSG	794
SASHELP	SHOES	395
SASHELP	SLKWXL	1,703
SASHELP	SMEMSG	34
SASHELP	SPRINGS	1,587
SASHELP	STEEL	44
SASHELP	STOCKS	699
SASHELP	STTMSG	336
SASHELP	SVRTDIST	2,373
SASHELP	SYR1001	105
SASHELP	TABLE	8
SASHELP	THICK	75
SASHELP	TIMEDATA	40,330
SASHELP	TOURISM	29
SASHELP	USECON	252
SASHELP	VBPLAYRS	11
SASHELP	VERBMGR	19
SASHELP	VIDMSG	7
SASHELP	VOTE1980	3,107
SASHELP	WEBMSG	349
SASHELP	WORKERS	67
SASHELP	YR1001	126
SASHELP	YR111	126
SASHELP	ZHC	7,445
SASHELP	ZIPCODE	41,232
SASHELP	ZTC	18,161
SASHELP	_CMPIDX_	44

Using PROC PRINT with Style

Hecht (2011) describes the appearance of PROC PRINT output can be customized with colors, backgrounds, fonts, justifications, and other report components using styles. Styles can be specified for all destinations (e.g., RTF, PDF, HTML, Excel, etc.) except the Listing destination. In the next example, the SASHELP.CARS data set is sorted in ascending order by the variables Origin, Type, Make and MSRP; the HTML destination is opened with the HTMLBlue style selected for output; and background and foreground styles selected for the data, obs and total parts of the PROC PRINT report output.

PROC PRINT Code:

```
proc sort data=sashelp.Cars(keep=Origin Type Make MSRP)
          out=work.Cars_Sorted ;
  where MSRP < 20000 ;
  by Origin Type Make MSRP ;
run ;

ods HTML path="/folders/myfolders"  (url=none)
         file="PROC-PRINT-with-Style.html"
         style=HTMLBlue ;
proc print data=work.Cars_Sorted
  style (data)  = [background=Blue   foreground=white]
  style (obs)   = [background=red    foreground=white]
  style (total) = [background=yellow foreground=black] ;
  by Origin Type ;
  id Origin Type Make ;
  format msrp dollar12.0 ;
```

```
  sum MSRP ;
run ;
ods HTML close ;
```

PROC PRINT Results:

Origin	Type	Make	MSRP
Asia	Hybrid	Honda	$19,110

Origin	Type	Make	MSRP
Asia	SUV	Honda	$18,690
		Honda	$19,860
		Kia	$19,635
		Mitsubishi	$18,892
		Suzuki	$17,163
Asia	SUV		$94,240

Origin	Type	Make	MSRP
Asia	Sports	Hyundai	$18,739

Origin	Type	Make	MSRP
Asia	Truck	Mazda	$14,840
		Nissan	$19,479
		Toyota	$12,800
		Toyota	$16,495
Asia	Truck		$63,614

Origin	Type	Make	MSRP
Asia	Wagon	Kia	$11,905
		Mitsubishi	$17,495
		Scion	$14,165
		Suzuki	$16,497
		Toyota	$16,695
Asia	Wagon		$76,757
Asia			$974,429

Origin	Type	Make	MSRP
Europe	Sedan	MINI	$16,999
		MINI	$19,999
		Volkswagen	$18,715
		Volkswagen	$19,825
Europe	Sedan		$75,538

Origin	Type	Make	MSRP
Europe	Wagon	Volkswagen	$19,005
Europe			$94,543

Origin	Type	Make	MSRP
USA	Sports	Ford	$18,345

Origin	Type	Make	MSRP
USA	Truck	Chevrolet	$18,760
		Dodge	$17,630
		Ford	$14,385
		GMC	$16,530
USA	Truck		$67,305

Origin	Type	Make	MSRP
USA	Wagon	Ford	$17,475
		Pontiac	$17,045
USA	Wagon		$34,520
USA			$495,400
			$1,564,372

Using Available Memory with Hash Object Programming

Dorfman (2009) describes a SAS hash object as, “a high-performance look-up table residing completely in the DATA step memory.” Due to the costs and availability of memory resources in today’s computing environments, software vendors are doing everything they can to develop language constructs that capitalize on memory-resident operations. Dorfman further describes that, “The hash object is implemented via a Data Step Component Interface (DSCI), meaning that it is not a part of the DATA step proper. Rather, picture it as a black-box device you can manipulate from inside the DATA step to ask it for lightning-quick data storage and retrieval services.”

Lafler (2016) describes a SAS hash object as, “a data structure that contains an array of items that are used to map identifying values, known as keys (e.g., employee IDs), to their associated values (e.g., employee names or employee addresses). As implemented, a hash object in the SAS System is used as a DATA step construct and is not available to any SAS Procedures.” A hash object reads the contents of a data set into memory once allowing the SAS system to repeatedly access the data, as necessary. The contents of a hash object can be saved to a SAS data set (or table), but at the end of the DATA step the hash object and all its contents disappear. Since memory-based operations are typically faster than their disk-based counterparts, users often experience faster and more efficient table lookup, merge, sort and transpose operations.

Users with DATA step programming experience will find the hash object syntax relatively straight forward to learn and use. Available in all operating systems running SAS 9 or greater, the hash object is called using methods. The syntax for calling a method involves specifying the name of the user-assigned hash table, a dot (.), the desired method (e.g., operation) by name, and finally the specification for the method enclosed in parentheses. The following example illustrates the basic syntax for calling a method to define a key.

```
MatchTitles.DefineKey ('Title');
```

where MatchTitles is the name of the hash table, DefineKey is the name of the called method, and 'Title' is the specification being passed to the method.

An essential operation frequently performed by users is the process of table lookup or search. The hash object as implemented in the DATA step provides users with the necessary tools to conduct match-merges (or joins) of two or more data sets. Data does not have to be sorted (or be in a designated sort order) before use as it does with the DATA step merge process. The next example illustrates a hash object with a simple key (TITLE) to merge (or join) the MOVIES and ACTORS data sets to create a new data set (MATCH_ON_MOVIE_TITLES) with matched observations.

Hash Object Code:

```
data match_on_movie_titles(drop=rc) ;

❶ if 0 then set mydata.movies
                mydata.actors ;  /* load variable properties into hash tables */

  if _n_ = 1 then do ;
❷    declare Hash MatchTitles (dataset:'mydata.actors') ; /* declare the name
                                                          MatchTitles for hash */

❸    MatchTitles.DefineKey ('Title') ; /* identify variable to use as key */
     MatchTitles.DefineData ('Actor_Leading',
                             'Actor_Supporting') ; /* identify columns of data */
     MatchTitles.DefineDone () ; /* complete hash table definition */
  end ;

  set mydata.movies ;

❹ if MatchTitles.find(key:title) = 0 then output ; /* lookup TITLE in MOVIES table
                                                      using MatchTitles hash */
run ;
```

Hash Object Results:

	Title	Length	Category	Year	Studio	Rating	Actor_Leading	Actor_Supporting
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R	Mel Gibson	Sophie Marceau
2	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
3	Coming to America	116	Comedy	1988	Paramount Pictures	R	Eddie Murphy	Arsenio Hall
4	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13	Tom Hanks	Sally Field
5	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13	Patrick Swayze	Demi Moore
6	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R	Mel Gibson	Danny Glover
7	Michael	106	Drama	1997	Warner Brothers	PG-13	John Trevolta	Andie MacDowell
8	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13	Chevy Chase	Beverly D'Angelo
9	Rocky	120	Action Adventure	1976	MGM / UA	PG	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	118	Drama Suspense	1991	Orion	R	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG	Sean Connery	Alec Baldwin
12	The Terminator	108	Action Sci-Fi	1984	Live Entertainment	R	Arnold Schwarzenegge	Michael Biehn
13	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13	Leonardo DiCaprio	Kate Winslet

Conclusion

As SAS users around the world celebrate a milestone of more than 40-years using SAS software, users should learn as many essential programming techniques as possible to enhance their careers well into the 21st century and beyond. This paper identified and shared several essential programming techniques that a pragmatic user and programmer should learn. From topics related to conditional logic scenarios; subsetting with WHERE expressions; determining the number of by-group levels that exist within classification variables; data manipulation with the family of CAT functions; merging or joining multiple tables of data; performing table lookup operations with user-defined formats; creating single-value and value-list macro variables with PROC SQL; examining and processing the contents of value-list macro variables; determining the FIRST., LAST. and Between by-group rows; processing repetitive data with arrays; and using metadata to better understand the contents of SAS datasets.

References

Efficiency and Performance Tuning References and Suggested Reading

Brown, Tony and Margaret Crevar (2016). "*Architecting Your SAS Grid®: Networking for Performance*," Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Cohen, Robert A. and Robert N. Rodriguez (2013). "*High-Performance Statistical Modeling*," Proceedings of the 2013 SAS Global Forum (SGF) Conference.

Kaufmann, Shaun (2016). *"High-Performance Data Access with FedSQL and DS2,"* Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2016). "*Top Ten SAS® Performance Tuning Techniques*," Proceedings of the 2016 MidWest SAS Users Group (MWSUG) Conference.

Lavery, Russ (2013). "*Fast Access Tricks for Large Sorted SAS Files*," Proceedings of the 2013 MidWest SAS Users Group (MWSUG) Conference.

Lui, Lingqun (2017). "*SAS Advanced Programming with Efficiency in Mind: A Real Case Study*," Proceedings of the 2017 Michigan SAS Users Group (MISUG) Conference.

Warner-Freeman, Jennifer K. (2007). "*I Cut My Processing Time By 90% Using Hash Tables - You Can Do It Too!*," Proceedings of the 2007 North East SAS Users Group (NESUG) Conference.

Williams, Michael; Gretel Easter and Steve Bradsher (2009). *"Troubleshoot Your Performance Issues: SAS® Technical Support Shows You How,"* Proceedings of the 2009 SAS Global Forum (SGF) Conference.

Hash Object References and Suggested Reading

Burlew, Michele M. (2012), "*SAS® Hash Object Programming Made Easy*," SAS Press, SAS Institute, Cary, NC, USA.

Dorfman, Paul M. and Don Henderson (2017). "*Beyond Table Lookup: The Versatile SAS® Hash Object*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Dorfman, Paul M. (2016). "*Using the SAS® Hash Object with Duplicate Key Entries*," Proceedings of the 2016 SAS Global Forum (SGF) Conference.

Dorfman, Paul and Peter Eberhardt (2010). "*Two Guys on Hash*," Proceedings of the 2010 South East SAS Users Group (SESUG) Conference.

Dorfman, Paul (2009). "*The SAS® Hash Object in Action*," Proceedings of the 2009 South East SAS Users Group (SESUG) Conference.

Lafler, Kirk Paul (2016). "*An Introduction to SAS® Hash Programming Techniques*," Proceedings of the 2016 SouthEast SAS Users Group (SESUG) Conference.

Loren, Judy (2008). "*How Do I Love Hash Tables? Let Me Count The Ways!*," Proceedings of the 2008 SAS Global Forum (SGF) Conference.

Mazloom, Dari (2017). "*SAS Hash Objects, Demystified*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Sakya, Daniel (2012). "*SAS® HASH Programming Basics*," Proceedings of the 2012 South Central SAS Users Group (SCSUG) Education Forum / Conference.

Schacherer, Chris (2015). "*Introduction to SAS® Hash Objects*," Proceedings of the 2015 SAS Global Forum (SGF) Conference.

Secosky, Jason and Janice Bloom (2007). "*Getting Started with the DATA Step Hash Object*," Proceedings of the 2007 SAS Global Forum (SGF) Conference.

Warner-Freeman, Jennifer K. (2007). "*I Cut My Processing Time By 90% Using Hash Tables - You Can Do It Too!*," Proceedings of the 2007 North East SAS Users Group (NESUG) Conference.

Macro References and Suggested Reading

Carpenter, Art (2016). *Carpenter's Complete Guide to the SAS® Macro Language, Third Edition*, SAS Institute Inc., Cary, NC.

Lui, Lingqun (2007). "*Passing Data Set Values into Application Parameters*," Proceedings of the 2007 MidWest SAS Users Group (MWSUG) Conference.

Roberts, Clark (1997). *"Building and Using Macro Variable Lists,"* Proceedings of the 1997 SAS Users Group International (SUGI) Conference.

SAS Programming Techniques References and Suggested Reading

Benjamin, William E. Jr. (2012). *"Leave Your Bad Code Behind: 50 Ways to Make Your SAS® Code Execute More Efficiently,"* Proceedings of the 2012 SAS Global Forum (SGF) Conference.

Cassidy, Deb (2003). "*Keeping Up With the FUN: New Functions in SAS 9*," Proceedings of the 2003 SouthEast SAS Users Group Conference.

Cody, Ron (2012). "*A Survey of Some of the Most Useful SAS® Functions*," Proceedings of the 2012 SAS Global Forum (SGF) Conference.

Gupta, Sunil (2006). "*WHERE vs. IF Statements: Knowing the Difference in How and When to Apply*," Proceedings of the 2006 SAS Users Group International (SUGI) Conference.

Hecht, Darylene (2011). "*PROC PRINT and ODS: Teaching an Old PROC New Tricks*," Proceedings of the 2011 SAS Global Forum (SGF) Conference.

Horstman, Joshua M. (2017). "*Beyond IF THEN ELSE: Techniques for Conditional Execution of SAS® Code*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2017). "An Introduction to PROC REPORT," Proceedings of the 2017 South Central SAS Users Group (SCSUG) Education Forum / Conference.

Lafler, Kirk Paul (2017). "*Best Practice Programming Techniques for SAS® Users*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2017). "*Removing Duplicates Using SAS®*," Proceedings of the 2017 SAS Global Forum (SGF) Conference.

Lafler, Kirk Paul (2014). "*Conditional Processing Using the Case Expression in PROC SQL*," Proceedings of the 2014 South Central SAS Users Group (SCSUG) Education Forum / Conference.

Lafler, Kirk Paul (2013). *PROC SQL: Beyond the Basics Using SAS, Second Edition*, SAS Institute Inc., Cary, NC, USA.

Lafler, Kirk Paul (2009). "*SAS® Macro Programming Tips and Techniques*," Proceedings of the 20009 SAS Global Forum (SGF) Conference.

Lavery, Russ (2016). "*An Animated Guide: The Internals of PROC REPORT*," Proceedings of the 2016 MidWest SAS Users Group (MWSUG) Conference.

Lui, Lingqun (2007). "*Passing Data Set Values into Application Parameters*," Proceedings of the 2007 MidWest SAS Users Group (MWSUG) Conference.

Repole Jr, Warren (2009). *"Don't Be a SAS® Dinosaur: Modernizing Programs with Base SAS 9.2 Enhancements,"* Proceedings of the 2009 SAS Global Forum (SGF) Conference.

Riba, S. David (1996). *"Redesigning a Legacy: Techniques of a Quality Partner,"* Proceedings of the 1996 SAS Users Group International (SUGI) Conference.

Roberts, Clark; Deborah Testa and Russell Holmes (19997). *"Audit Trail Plug-ins for SAS® Software Applications,"* Proceedings of the 1999 Western Users of SAS Software (WUSS) Conference.

Roberts, Clark (1997). *"Building and Using Macro Variable Lists,"* Proceedings of the 1997 SAS Users Group International (SUGI) Conference.

Shapiro, Mira (2016). "*SAS® Functions You May Have Been MISSING*," Proceedings of the 2016 PharmaSUG Conference.

Sun, GuanGhui (Brian) (2011). "*Why Dummy Variable Makes You SMART, and How to Do it SEXY*," Proceedings of the 2011 Western Users of SAS Software (WUSS) Conference.

Venam, Srinivas; Manvitha Yennam; and Phaneendhar Vanam (2016). "*Good Programming Practice [GPP] in SAS® & Clinical Trials*," Proceedings of the 2016 Western Users of SAS Software (WUSS) Conference.

Wang, Hui (2015). "*Creating Data-Driven SAS® Code with CALL EXECUTE*," Proceedings of the 2015 PharmaSUG Conference.

Whitlock, Ian (2006). "How to Think Through the SAS® DATA Step," Proceedings of the 2006 SAS Users Group International (SUGI) Conference.

Whitlock, Ian (1998). "*CALL EXECUTE: How and Why*," Proceedings of the 1998 SAS Users Group International (SUGI) Conference.

Acknowledgments

The author thanks Yunin Ludena, Beginners Techniques Section Chair, for accepting my abstract and paper; Nina Worden, WUSS 2018 Academic Program Chair; Andra Northup, WUSS 2018 Operations Chair; the Western Users of SAS Software (WUSS) Executive Board; and SAS Institute for organizing and supporting a great conference!

Trademark Citations

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.

Data Sets Used in Examples

The examples presented in this paper include the RUGs_2015 and RUGs_2016 data sets; and several in the SASHELP library including the CARS, and HEART data sets. You'll be able to access and use these data sets for example purposes and for testing the enclosed code examples.

The RUGs_2015 data set consists of 4 observations and 3 variables, illustrated below.

	RUG	Number_Papers	Year
1	MWSUG	96	2015
2	SCSUG	28	2015
3	SESUG	140	2015
4	WUSS	102	2015

Data Set #1. RUGs_2015

The RUGs_2016 data set consists of 4 observations and 3 variables, illustrated below.

	RUG	Number_Papers	Year
1	MWSUG	124	2016
2	SCSUG	62	2016
3	SESUG	140	2016
4	WUSS	112	2016

Data Set #2. RUGs_2016

The MOVIES data set consists of 22 observations and 6 variables, illustrated below.

	Title	Length	Category	Year	Studio	Rating
1	Brave Heart	177	Action Adventure	1995	Paramount Pictures	R
2	Casablanca	103	Drama	1942	MGM / UA	PG
3	Christmas Vacation	97	Comedy	1989	Warner Brothers	PG-13
4	Coming to America	116	Comedy	1988	Paramount Pictures	R
5	Dracula	130	Horror	1993	Columbia TriStar	R
6	Dressed to Kill	105	Drama Mysteries	1980	Filmways Pictures	R
7	Forrest Gump	142	Drama	1994	Paramount Pictures	PG-13
8	Ghost	127	Drama Romance	1990	Paramount Pictures	PG-13
9	Jaws	125	Action Adventure	1975	Universal Studios	PG
10	Jurassic Park	127	Action	1993	Universal Pictures	PG-13
11	Lethal Weapon	110	Action Cops & Robber	1987	Warner Brothers	R
12	Michael	106	Drama	1997	Warner Brothers	PG-13
13	National Lampoon's Vacation	98	Comedy	1983	Warner Brothers	PG-13
14	Poltergeist	115	Horror	1982	MGM / UA	PG
15	Rocky	120	Action Adventure	1976	MGM / UA	PG
16	Scarface	170	Action Cops & Robber	1983	Universal Studios	R
17	Silence of the Lambs	118	Drama Suspense	1991	Orion	R
18	Star Wars	124	Action SciFi	1977	Lucas Film Ltd	PG
19	The Hunt for Red October	135	Action Adventure	1989	Paramount Pictures	PG
20	The Terminator	108	Action SciFi	1984	Live Entertainment	R
21	The Wizard of Oz	101	Adventure	1939	MGM / UA	G
22	Titanic	194	Drama Romance	1997	Paramount Pictures	PG-13

Data Set #3. MOVIES

The ACTORS data set consists of 13 observations and 3 variables, illustrated below.

	Title	Actor_Leading	Actor_Supporting
1	Brave Heart	Mel Gibson	Sophie Marceau
2	Christmas Vacation	Chevy Chase	Beverly D'Angelo
3	Coming to America	Eddie Murphy	Arsenio Hall
4	Forrest Gump	Tom Hanks	Sally Field
5	Ghost	Patrick Swayze	Demi Moore
6	Lethal Weapon	Mel Gibson	Danny Glover
7	Michael	John Travolta	Andie MacDowell
8	National Lampoon's Vacation	Chevy Chase	Beverly D'Angelo
9	Rocky	Sylvester Stallone	Talia Shire
10	Silence of the Lambs	Anthony Hopkins	Jodie Foster
11	The Hunt for Red October	Sean Connery	Alec Baldwin
12	The Terminator	Arnold Schwarzenegger	Michael Biehn
13	Titanic	Leonardo DiCaprio	Kate Winslet

Data Set #4. ACTORS

The SASHELP.CARS data set consists of 428 observations and 15 variables, illustrated below.

	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	EngineSize	Cylinders	Horsepower	MPG_City	MPG_Highway	Weight	Wheelbase	Length
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337	3.5	6	265	17	23	4451	106	189
2	Acura	RSX Type S 2dr	Sedan	Asia	Front	$23,820	$21,761	2	4	200	24	31	2778	101	172
3	Acura	TSX 4dr	Sedan	Asia	Front	$26,990	$24,647	2.4	4	200	22	29	3230	105	183
4	Acura	TL 4dr	Sedan	Asia	Front	$33,195	$30,299	3.2	6	270	20	28	3575	108	186
5	Acura	3.5 RL 4dr	Sedan	Asia	Front	$43,755	$39,014	3.5	6	225	18	24	3880	115	197
6	Acura	3.5 RL w/Navigation 4dr	Sedan	Asia	Front	$46,100	$41,100	3.5	6	225	18	24	3893	115	197
7	Acura	NSX coupe 2dr manual S	Sports	Asia	Rear	$89,765	$79,978	3.2	6	290	17	24	3153	100	174
8	Audi	A4 1.8T 4dr	Sedan	Europe	Front	$25,940	$23,508	1.8	4	170	22	31	3252	104	179
9	Audi	A4 1.8T convertible 2dr	Sedan	Europe	Front	$35,940	$32,506	1.8	4	170	23	30	3638	105	180
10	Audi	A4 3.0 4dr	Sedan	Europe	Front	$31,840	$28,846	3	6	220	20	28	3462	104	179
11	Audi	A4 3.0 Quattro 4dr manual	Sedan	Europe	All	$33,430	$30,366	3	6	220	17	26	3583	104	179
12	Audi	A4 3.0 Quattro 4dr auto	Sedan	Europe	All	$34,480	$31,388	3	6	220	18	25	3627	104	179
13	Audi	A6 3.0 4dr	Sedan	Europe	Front	$36,640	$33,129	3	6	220	20	27	3561	109	192
14	Audi	A6 3.0 Quattro 4dr	Sedan	Europe	All	$39,640	$35,992	3	6	220	18	25	3880	109	192
15	Audi	A4 3.0 convertible 2dr	Sedan	Europe	Front	$42,490	$38,325	3	6	220	20	27	3814	105	180
16	Audi	A4 3.0 Quattro convertible 2dr	Sedan	Europe	All	$44,240	$40,075	3	6	220	18	25	4013	105	180
17	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	Europe	All	$42,840	$38,840	2.7	6	250	18	25	3836	109	192
18	Audi	A6 4.2 Quattro 4dr	Sedan	Europe	All	$49,690	$44,936	4.2	8	300	17	24	4024	109	193
19	Audi	A8 L Quattro 4dr	Sedan	Europe	All	$69,190	$64,740	4.2	8	330	17	24	4399	121	204
20	Audi	S4 Quattro 4dr	Sedan	Europe	All	$48,040	$43,556	4.2	8	340	14	20	3825	104	179
21	Audi	RS 6 4dr	Sports	Europe	Front	$84,600	$76,417	4.2	8	450	15	22	4024	109	191
22	Audi	TT 1.8 convertible 2dr (coupe)	Sports	Europe	Front	$35,940	$32,512	1.8	4	180	20	28	3131	95	159
23	Audi	TT 1.8 Quattro 2dr (convertible)	Sports	Europe	All	$37,390	$33,891	1.8	4	225	20	28	2921	96	159
24	Audi	TT 3.2 coupe 2dr (convertible)	Sports	Europe	All	$40,590	$36,739	3.2	6	250	21	29	3351	96	159
25	Audi	A6 3.0 Avant Quattro	Wagon	Europe	All	$40,840	$37,060	3	6	220	18	25	4035	109	192
26	Audi	S4 Avant Quattro	Wagon	Europe	All	$49,090	$44,446	4.2	8	340	15	21	3936	104	179
27	BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873	3	6	225	16	23	4023	110	180
28	BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720	4.4	8	325	16	22	4824	111	184

Data Set #5. SASHELP.CARS

The SASHELP.HEART data set consists of 5,209 observations and 17 variables, illustrated below.

	Status	DeathCause	AgeCHDdiag	Sex	AgeAtStart	Height	Weight	Diastolic	Systolic	MRW	Smoking	AgeAtDeath	Cholesterol	Chol_Status	BP_Status	Weight_Status	Smoking_Status
1	Dead	Other		Female	29	62.5	140	78	124	121	0	55			Normal	Overweight	Non-smoker
2	Dead	Cancer		Female	41	59.75	194	92	144	183	0	57	181	Desirable	High	Overweight	Non-smoker
3	Alive			Female	57	62.25	132	90	170	114	10		250	High	High	Overweight	Moderate (6-15)
4	Alive			Female	39	65.75	158	80	128	123	0		242	High	Normal	Overweight	Non-smoker
5	Alive			Male	42	66	156	76	110	116	20		281	High	Optimal	Overweight	Heavy (16-25)
6	Alive			Female	58	61.75	131	92	176	117	0		196	Desirable	High	Overweight	Non-smoker
7	Alive			Female	36	64.75	136	80	112	110	15		156	Desirable	Normal	Overweight	Moderate (6-15)
8	Dead	Other		Male	53	65.5	130	80	114	99	0	77	276	High	Normal	Normal	Non-smoker
9	Alive			Male	35	71	194	68	132	124	0		211	Borderline	Normal	Overweight	Non-smoker
10	Dead	Cerebral Vascular Disease		Male	52	62.5	129	78	124	106	5	82	284	High	Normal	Normal	Light (1-5)
11	Alive			Male	39	66.25	179	76	128	133	30		225	Borderline	Normal	Overweight	Very Heavy (> 25)
12	Alive		57	Male	33	64.25	151	68	108	118	0		221	Borderline	Optimal	Overweight	Non-smoker
13	Alive		55	Male	33	70	174	90	142	114	0		188	Desirable	High	Overweight	Non-smoker
14	Alive		79	Male	57	67.25	165	76	128	118	15				Normal	Overweight	Moderate (6-15)
15	Alive		66	Male	44	69	155	90	130	105	30		292	High	High	Normal	Very Heavy (> 25)
16	Alive			Female	37	64.5	134	76	120	108	10		196	Desirable	Normal	Normal	Moderate (6-15)
17	Alive			Male	40	66.25	151	72	132	112	30		192	Desirable	Normal	Overweight	Very Heavy (> 25)
18	Dead	Cancer	56	Male	56	67.25	122	72	120	87	15	72	194	Desirable	Normal	Underweight	Moderate (6-15)
19	Alive			Female	42	67.75	162	96	138	119	1		200	Borderline	High	Overweight	Light (1-5)
20	Dead	Coronary Heart Disease	74	Male	46	66.5	157	84	142	116	30	76	233	Borderline	High	Overweight	Very Heavy (> 25)
21	Alive			Female	37	66.25	148	78	110	112	15		192	Desirable	Optimal	Overweight	Moderate (6-15)
22	Alive			Female	45	64	147	74	120	119	5		209	Borderline	Normal	Overweight	Light (1-5)
23	Alive			Female	59	65.75	156	74	156	122	0		200	Borderline	High	Overweight	Non-smoker
24	Alive			Female	36	63.75	122	84	132	102	0		184	Desirable	Normal	Normal	Non-smoker
25	Alive			Female	50	67.5	185	88	150	136	15		228	Borderline	High	Overweight	Moderate (6-15)

Data Set #6. SASHELP.HEART

Author Bio

Kirk Paul Lafler is an entrepreneur, consultant and founder at Software Intelligence Corporation, and has been using SAS since 1979. Kirk is a SAS application developer, programmer, certified professional, provider of SAS consulting and training services, mentor, advisor and adjunct professor at University of California San Diego Extension, emeritus sasCommunity.org Advisory Board member, and educator to SAS users around the world. As the author of six books including Google® Search Complete (Odyssey Press. 2014) and PROC SQL: Beyond the Basics Using SAS, Second Edition (SAS Press. 2013); Kirk has written hundreds of papers and articles; served as an Invited speaker, trainer, keynote and section leader at SAS conferences and meetings around the world; and is the recipient of 25 “Best” contributed paper, hands-on workshop (HOW), and poster awards.

Comments and suggestions can be sent to:

Kirk Paul Lafler
SAS® Consultant, Application Developer, Programmer, Data Analyst, Educator and Author
Software Intelligence Corporation
E-mail: KirkLafler@cs.com
LinkedIn: http://www.linkedin.com/in/KirkPaulLafler
Twitter: @sasNerd