Paper 2685-2015

How Do You Use Look-up Tables?

Philip R Holland, Holland Numerics Limited

ABSTRACT

No matter what type of programming you do in a pharmaceutical environment there will eventually be a need to combine your data with a look-up table. This look-up table could be a code list for adverse events, a list of names for visits, or just one of your own summary data sets containing totals that you will be using to calculate percentages, and you may have your favourite way to incorporate it. This paper will describe, and discuss the reasons for, using 5 different simple ways to merge data sets with look-up tables, so that, when you take over the maintenance of a new program, you will be ready for anything!

INTRODUCTION

All 5 techniques described in this paper use the same 4 SAS data sets to create the same output data set.

Sample Data Sets

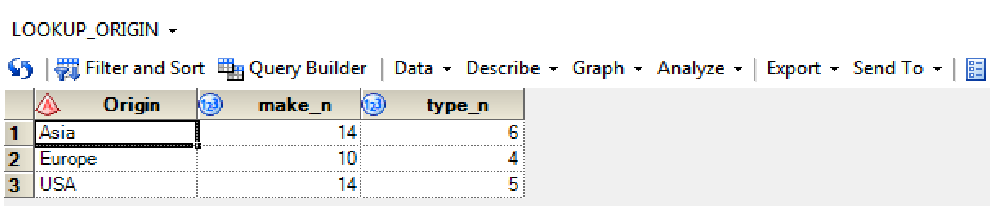
The SAS data sets used in this paper are:

- MAIN = Multiple copies of SASHELP.CARS (428 observations and 15 variables) saved in a single WORK data set to increase the size of this data set.

```
%LET mult = 1;  /* 10, 100, 1000, 2000, 5000 */

DATA main;
    SET sashelp.cars;
    DO i = 1 TO &mult.;
        OUTPUT;
    END;
RUN;
```

- LOOKUP_ORIGIN (3 observations and 3 variables):

```
PROC SQL;
    CREATE TABLE lookup_origin AS
        SELECT origin  /* key */
              ,COUNT(DISTINCT make) AS make_n
              ,COUNT(DISTINCT type) AS type_n
        FROM   sashelp.cars
        GROUP BY
               origin
        ORDER BY
               origin
    ;
QUIT;
```



	Origin	make_n	type_n
1	Asia	14	6
2	Europe	10	4
3	USA	14	5

- LOOKUP_TYPE (15 observations and 6 variables):

```
PROC SQL;
    CREATE TABLE lookup_make AS
        SELECT origin, make  /* keys */
              ,COUNT(DISTINCT model) AS make_model_n
              ,COUNT(DISTINCT type) AS make_type_n
              ,MEAN(msrp) AS make_msrp_mean
              ,MAX(horsepower) AS make_horsepower_max
        FROM   sashelp.cars
        GROUP BY
               origin, make
        ORDER BY
               origin, make
    ;
QUIT;
```

LOOKUP_TYPE

Filter and Sort | Query Builder | Data | Describe | Graph | Analyze | Export | Send To

	Origin	Type	type_model_n	type_make_n	type_msrp_mean	type_horsepower_max
1	Asia	Hybrid	3	2	19920	110
2	Asia	SUV	25	11	29569	325
3	Asia	Sedan	93	13	22763.968085	340
4	Asia	Sports	17	9	32510.647059	300
5	Asia	Truck	8	4	20383.625	305
6	Asia	Wagon	11	9	23143.727273	315
7	Europe	SUV	10	6	48346	340
8	Europe	Sedan	76	8	42992.051282	493
9	Europe	Sports	23	5	71998.695652	493
10	Europe	Wagon	12	6	37851.25	340
11	USA	SUV	25	12	34589.2	325
12	USA	Sedan	90	12	25638.833333	302
13	USA	Sports	9	6	45257.222222	500
14	USA	Truck	16	5	27220.25	345
15	USA	Wagon	7	6	22345.714286	250

- LOOKUP_MAKE (38 observations and 6 variables):

```
PROC SQL;
    CREATE TABLE lookup_type AS
        SELECT origin, type  /* keys */
              ,COUNT(DISTINCT model) AS type_model_n
              ,COUNT(DISTINCT make) AS type_make_n
              ,MEAN(msrp) AS type_msrp_mean
              ,MAX(horsepower) AS type_horsepower_max
        FROM   sashelp.cars
        GROUP BY
               origin, type
        ORDER BY
               origin, type
    ;
QUIT;
```

LOOKUP_MAKE

Filter and Sort | Query Builder | Data | Describe | Graph | Analyze | Export | Send To

	Origin	Make	make_model_n	make_type_n	make_msrp_mean	make_horsepower_max
1	Asia	Acura	7	3	42938.571429	290
2	Asia	Honda	17	4	21434.705882	240
3	Asia	Hyundai	12	3	17476.5	194
4	Asia	Infiniti	7	2	36070	340
5	Asia	Isuzu	2	1	26149	275
6	Asia	Kia	11	3	15875.909091	195
7	Asia	Lexus	11	4	44215.454545	300
8	Asia	Mazda	11	4	21770.727273	238
9	Asia	Mitsubishi	13	4	23423.615385	271
10	Asia	Nissan	17	5	24730.941176	305
11	Asia	Scion	2	2	13565	108
12	Asia	Subaru	11	4	25501.818182	300
13	Asia	Suzuki	8	3	16230.25	185
14	Asia	Toyota	28	6	22524.464286	325
15	Europe	Audi	19	3	43307.894737	450
16	Europe	BMW	20	4	43285.25	333
17	Europe	Jaguar	12	2	61580.416667	390
18	Europe	Land Rover	3	1	45831.666667	282
19	Europe	MINI	2	1	18499	163
20	Europe	Mercedes-Benz	24	4	60656.807692	493
21	Europe	Porsche	7	2	83565	477
22	Europe	Saab	7	2	37640	250
23	Europe	Volkswagen	15	3	32248.666667	420
24	Europe	Volvo	12	3	36314.166667	300
25	USA	Buick	9	2	30537.777778	275
26	USA	Cadillac	8	4	50474.375	345
27	USA	Chevrolet	27	5	26587.037037	350
28	USA	Chrysler	15	3	27252	255
29	USA	Dodge	13	4	26253.846154	500
30	USA	Ford	23	5	24015.869565	310
31	USA	GMC	8	3	29560.5	325

DATA STEP MERGE

This is probably the most commonly used technique to merge SAS data sets together. However, it is also one of the least efficient method, as, for each join, the 2 data sets must be sorted the same way. In this example, even though the smaller lookup data sets would be quicker to sort, the large master data set has to resorted each time.

The output data set will include 4 new calculated variables:

- make_msrp_flag: which is set to 1 if msrp > mean msrp by make.
- make_horsepower_pct: which is set to the percentage of the maximum horsepower by make.
- type_msrp_flag: which is set to 1 if msrp > mean msrp by type.
- type_horsepower_pct: which is set to the percentage of the maximum horsepower by type.

```
PROC SORT DATA = main OUT = datastepmerge1;
    BY origin make;
RUN;

DATA datastepmerge2;
    MERGE datastepmerge1 lookup_origin;
    BY origin;
RUN;

DATA datastepmerge3;
    MERGE datastepmerge2 lookup_make;
    BY origin make;
    IF msrp > make_msrp_mean THEN make_msrp_flag = 1;
                             ELSE make_msrp_flag = 0;
    make_horsepower_pct = 100 * horsepower / make_horsepower_max;
RUN;
```

```
PROC SORT DATA = datastepmerge3 OUT = datastepmerge4;
    BY origin type;
RUN;

DATA datastepmerge5;
    MERGE datastepmerge4 lookup_type;
    BY origin type;
    IF msrp > type_msrp_mean THEN type_msrp_flag = 1;
                             ELSE type_msrp_flag = 0;
    type_horsepower_pct = 100 * horsepower / type_horsepower_max;
RUN;
```

SQL JOIN

The strange fact about PROC SQL is that it becomes less efficient with increasing data, and yet it is rarely used with clinical data, where the data volumes are low, but widely used with financial data, where the data volumes are high.

```
PROC SQL;
    CREATE TABLE sqljoin1 AS
        SELECT a.*
              ,b.make_n
              ,b.type_n
              ,c.make_model_n
              ,c.make_type_n
              ,c.make_msrp_mean
              ,(CASE
                WHEN a.msrp > c.make_msrp_mean THEN 1
                ELSE 0
                END) AS make_msrp_flag
              ,c.make_horsepower_max
              ,(100 * a.horsepower / c.make_horsepower_max)
                 AS make_horsepower_pct
              ,d.type_model_n
              ,d.type_make_n
              ,d.type_msrp_mean
              ,(CASE
                WHEN a.msrp > d.type_msrp_mean THEN 1
                ELSE 0
                END) AS type_msrp_flag
              ,d.type_horsepower_max
              ,(100 * a.horsepower / d.type_horsepower_max)
                 AS type_horsepower_pct
        FROM   main a
        LEFT JOIN
               lookup_origin b
        ON     a.origin = b.origin
        LEFT JOIN
               lookup_make c
        ON     a.origin = c.origin AND a.make = c.make
        LEFT JOIN
               lookup_type d
        ON     a.origin = d.origin AND a.type = d.type
    ;
QUIT;
```

GENERATED SAS FORMATS

Using SAS formats is inherently more efficient that joining data sets directly, as the format data is stored in memory, rather than on disk. There is a small downside, as you have to convert the data sets into formats, but as these data sets are relatively small, there is a significant benefit to using SAS formats as lookup tables. The available memory is going to be a limiting factor in the usable size of the format, but formats in excess of 50,000 entries are perfectly acceptable.

```
DATA format_origin;
    LENGTH fmtname $7 start $80 label 8 type hlo $1;
    SET lookup_origin;
    type = 'I';
    hlo = ' ';
    start = origin;
    fmtname = 'originm';
    label = make_n;
    output;
    fmtname = 'origint';
    label = type_n;
    output;
RUN;

PROC SORT DATA = format_origin NODUPKEY;
    BY fmtname start;
RUN;

PROC FORMAT CNTLIN = format_origin;
RUN;

%MACRO generate_format(level1=, level2=);
    DATA format_&level1.;
        LENGTH fmtname $7 start $80 label 8 type hlo $1;
        SET lookup_&level1. (RENAME = (&level1.=level1));
        type = 'I';
        hlo = ' ';
        start = CATX('|', origin, level1);
        fmtname = "&level1.c";
        label = &level1._model_n;
        output;
        fmtname = "&level1.x";
        label = &level1._&level2._n;
        output;
        fmtname = "&level1.p";
        label = &level1._msrp_mean;
        output;
        fmtname = "&level1.h";
        label = &level1._horsepower_max;
        output;
    RUN;

    PROC SORT DATA = format_&level1. NODUPKEY;
        BY fmtname start;
    RUN;

    PROC FORMAT CNTLIN = format_&level1.;
    RUN;
%MEND generate_format;
```

```
%generate_format(level1=make, level2=type);
%generate_format(level1=type, level2=make);

DATA format1;
    SET main;
    make_n = INPUT(origin, originm.);
    type_n = INPUT(origin, origint.);
    make_model_n = INPUT(CATX('|', origin, make), makec.);
    make_type_n = INPUT(CATX('|', origin, make), makex.);
    make_msrp_mean = INPUT(CATX('|', origin, make), makep.);
    IF msrp > make_msrp_mean THEN make_msrp_flag = 1;
                             ELSE make_msrp_flag = 0;
    make_horsepower_max = INPUT(CATX('|', origin, make), makeh.);
    make_horsepower_pct = 100 * horsepower / make_horsepower_max;
    type_model_n = INPUT(CATX('|', origin, type), typec.);
    type_make_n = INPUT(CATX('|', origin, type), typex.);
    type_msrp_mean = INPUT(CATX('|', origin, type), typep.);
    IF msrp > type_msrp_mean THEN type_msrp_flag = 1;
                             ELSE type_msrp_flag = 0;
    type_horsepower_max = INPUT(CATX('|', origin, type), typeh.);
    type_horsepower_pct = 100 * horsepower / type_horsepower_max;
RUN;
```

GENERATED IF .. THEN .. ELSE

To save all of the sorting and memory usage why not just generate Data Step code to add the extra information from the lookup data sets. In this case IF .. THEN .. ELSE statements are generated from the lookup data sets, and stored as text records in a SAS Catalog Source entry in a WORK catalog, so they are deleted automatically at the end of the SAS session.

```
FILENAME srcif CATALOG "work.generateif";

DATA _NULL_;
    SET lookup_origin END = eof;
    FILE srcif(origin.source);
    IF _N_ = 1 THEN PUT "IF origin = '" origin +(-1) "' THEN DO;";
    ELSE PUT "ELSE IF origin = '" origin +(-1) "' THEN DO;";
    PUT "make_n = " make_n ";";
    PUT "type_n = " type_n ";";
    PUT "END;";
RUN;

%MACRO generate_if(level1=, level2=);
    DATA _NULL_;
        SET lookup_&level1. END = eof;
        FILE srcif(&level1..source);
        IF _N_ = 1 THEN PUT "IF origin = '" origin +(-1)
                            "' AND &level1. = '"
                            &level1. +(-1) "' THEN DO;";
        ELSE PUT "ELSE IF origin = '" origin +(-1) "' AND &level1. = '"
                 &level1. +(-1) "' THEN DO;";
        PUT "&level1._model_n = " &level1._model_n ";";
        PUT "&level1._&level2._n = " &level1._&level2._n ";";
        PUT "&level1._msrp_mean = " &level1._msrp_mean ";";
        PUT "IF msrp > &level1._msrp_mean THEN &level1._msrp_flag = 1;";
        PUT "                              ELSE &level1._msrp_flag = 0;";
```

```
        PUT "&level1._horsepower_max = " &level1._horsepower_max ";";
        PUT "&level1._horsepower_pct = 100 * horsepower"
            " / &level1._horsepower_max;";
        PUT "END;";
    RUN;
%MEND generate_if;

%generate_if(level1=make, level2=type);
%generate_if(level1=type, level2=make);

DATA generateif1;
    SET main;
    %INCLUDE srcif(origin.source);
    %INCLUDE srcif(make.source);
    %INCLUDE srcif(type.source);
RUN;
```

GENERATED SELECT .. WHEN .. OTHERWISE

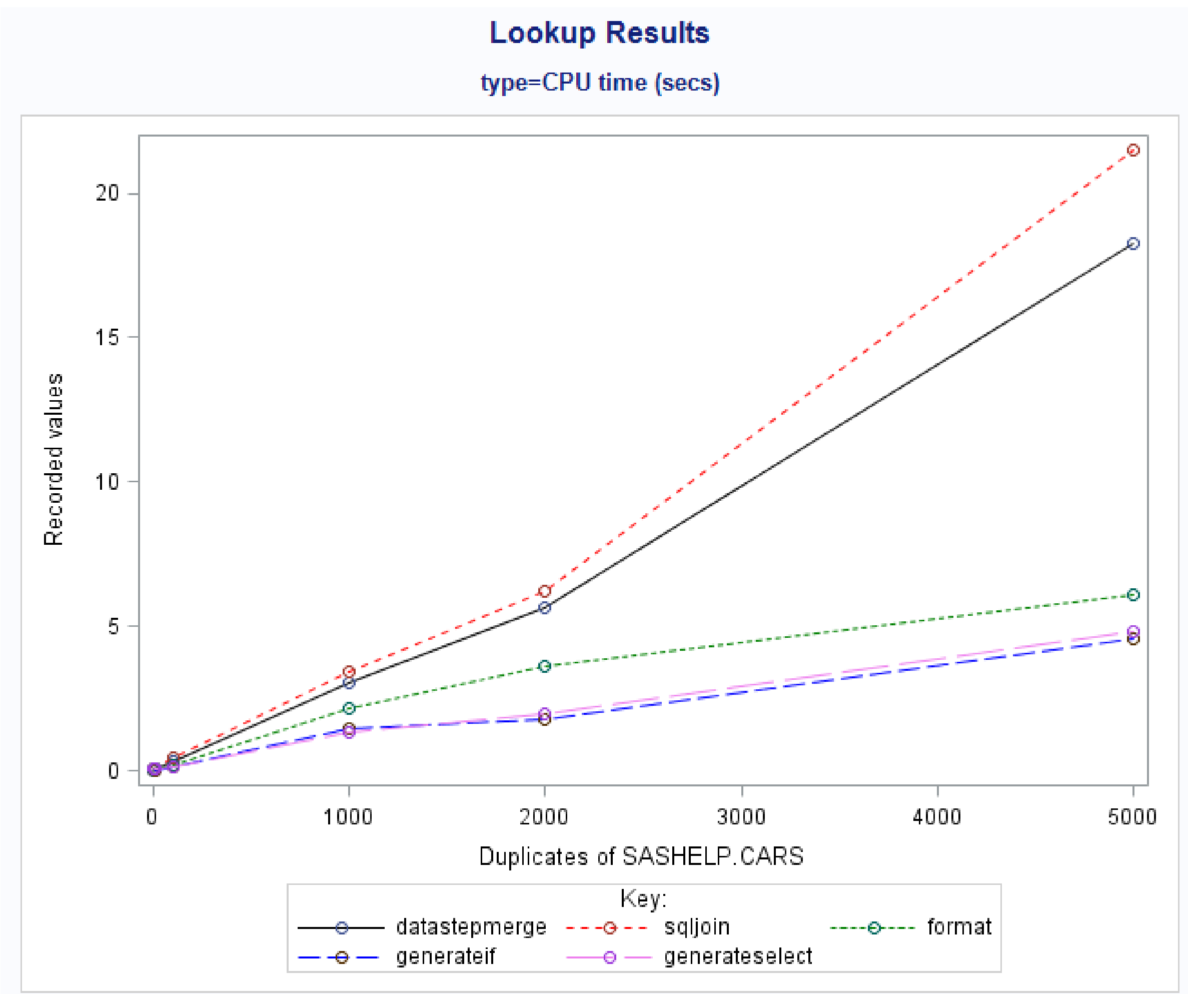
In this case SELECT .. WHEN .. OTHERWISE statements are generated from the lookup data sets, and stored as text records in a SAS Catalog Source entry in a WORK catalog, so they are deleted automatically at the end of the SAS session. The advantage of these statements over IF .. THEN .. ELSE is that the OTHERWISE statement forces an action if none of the previous test are satisfied, so it can be used to highlight any omissions.

```
FILENAME srcsel CATALOG "work.generateselect";

DATA _NULL_;
    SET lookup_origin END = eof;
    FILE srcsel(origin.source);
    IF _N_ = 1 THEN PUT "SELECT;";
    PUT "WHEN (origin = '" origin +(-1) "') DO;";
    PUT "make_n = " make_n ";";
    PUT "type_n = " type_n ";";
    PUT "END;";
    IF eof THEN DO;
        PUT "OTHERWISE;";
        PUT "END;";
    END;
RUN;

%MACRO generate_select(level1=, level2=);
    DATA _NULL_;
        SET lookup_&level1. END = eof;
        FILE srcsel(make.source);
        IF _N_ = 1 THEN PUT "SELECT;";
        PUT "WHEN (origin = '" origin +(-1) "' AND &level1. = '"
            &level1. +(-1) "') DO;";
        PUT "&level1._model_n = " &level1._model_n ";";
        PUT "&level1._&level2._n = " &level1._&level2._n ";";
        PUT "&level1._msrp_mean = " &level1._msrp_mean ";";
        PUT "IF msrp > &level1._msrp_mean THEN &level1._msrp_flag = 1;";
        PUT "                              ELSE &level1._msrp_flag = 0;";
        PUT "&level1._horsepower_max = " &level1._horsepower_max ";";
        PUT "&level1._horsepower_pct = 100 * horsepower"
            " / &level1._horsepower_max;";
        PUT "END;";
```

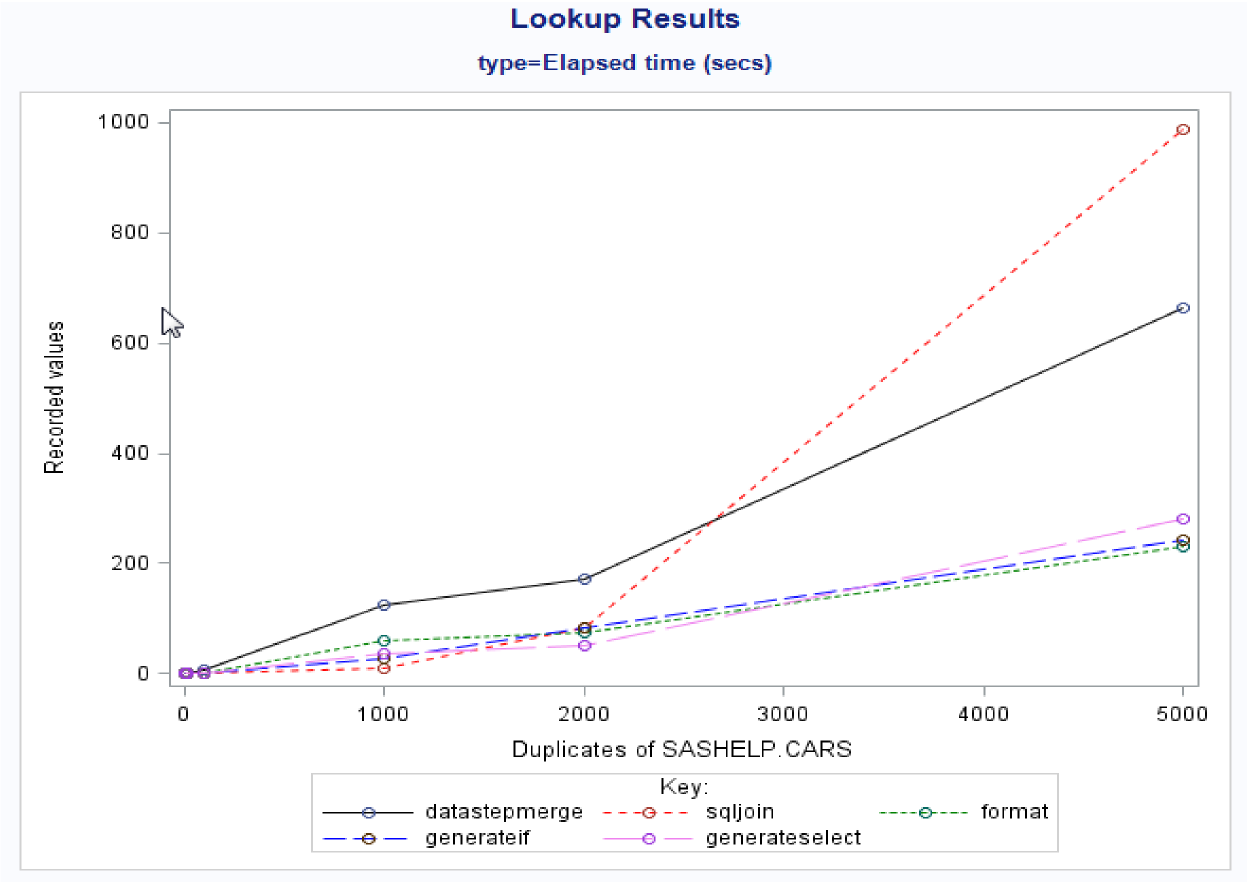
```
        IF eof THEN DO;
            PUT "OTHERWISE;";
            PUT "END;";
        END;
    RUN;
%MEND compare_select;

%generate_select(level1=make, level2=type);
%generate_select(level1=type, level2=make);

DATA generateselect1;
    SET main;
    %INCLUDE srcsel(origin.source);
    %INCLUDE srcsel(make.source);
    %INCLUDE srcsel(type.source);
RUN;
```

CONCLUSIONS

Comparing techniques using CPU time shows that Data Step Merge and SQL Joins are comparable, Formats are quicker, and the Data Step statement-generating techniques are the quickest.



However, comparing techniques using elapsed time is more interesting, as all of the techniques show a linear increase in elapsed time, apart from SQL Join, which increases dramatically as the volume increases after being fairly fast as low data volumes. This is because, at low data volumes, PROC SQL carries out most of its data processing in memory. At higher data volumes it is forced to use the WORK library to store intermediate data, which is much less efficient. Both Data Step Merge and SQL Join are significantly less efficient than the other 3 techniques, which minimise the number of data passes.



Are you now considering a new technique for using lookup tables?

REFERENCES

Holland, Philip R. June 2015. *SAS Programming and Data Visualization Techniques.* Apress.

CONTACT DETAILS

Your comments and questions are valued and encouraged. Contact the author at:

Author Name	Philip R Holland
Company	Holland Numerics Ltd
Address	94 Green Drift
City / Postcode	Royston, Hertfordshire, SG8 5BT, United Kingdom
Work Phone:	+44 7714 279085
Fax:	+44 1763 242486
Email:	phil@hollandnumerics.com
Web:	www.hollandnumerics.com/SASPAPER.HTM

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.