SAS ® Program Efficiency for Beginners

Bruce Gilsen

Federal Reserve Board

October 2023

Introduction

- How tips are normally presented
  - Describe a technique
  - Give examples of technique
- Identification problem
- How this paper presents tips
  - Describe application or data set
  - Present techniques for application or data set
  - Simple examples, small programs

Summary of Programming Tips

- Read flat file, keep selected observations
- Subset SAS data set, create new SAS data set
- Subset SAS data set to use in PROC step
- Test multiple conditions with IN, OR, or AND operators
- Subset SAS data set with WHERE statement
- Read SAS data set, need only a few variables
- Concatenate one SAS data set to another
- Execute DATA step, output SAS data set not needed
- In DATA step, denominator of division could be zero

Digressions

- What is a SAS data set?
- What is an observation and a variable?
- What is the SAS Program Data Vector (PDV)?
- What is a subsetting IF statement?
  - Test a condition
  - If condition is true, continue processing observation
  - If condition is false, discard observation
  - Different than IF-THEN/ELSE statement

1. Read flat file, long records, keep selected observations based on a few variables

- Less efficient
  - Read entire record
  - Check variable values
  - Keep or discard record

- More efficient
  - Read only variables needed to decide if should keep record
  - Check variable values
  - Use trailing @ to hold record for possible re-read
  - Discard record or read rest of record

Method 1, less efficient

```
data income;
  infile salesdata;
  input @1  shoetype 8.
        @9  netinc 8.
        @17 nextvar 8.
                .
                .
                .
       @1993 lastvar 8.;
  if netinc > 100;
run;
```

Method 2, more efficient

```
data income;
  infile salesdata;
  input @9 netinc 8. @;
  if netinc > 100;
  input @1 shoetype 8.
        @17 nextvar 8.
                .
                .
                .
        @1993 lastvar 8.;
run;
```

Simpler code if all fields have same informat

```
Method 2

data income;
  infile salesdata;
  input @9  netinc 8. @;
  if netinc > 100;
  input @1 shoetype 8.
        @17 nextvar 8.
                .
                .
                .
       @1993 lastvar 8.;
run;
```

```
Method 2, simpler

data income;
  infile salesdata;
  input @9 netinc 8. @;
  if netinc > 100;
  input @1 shoetype 8.
        @17
        (nextvar
             .
        lastvar)
        (8.);
run;
```

2. Subset SAS data set, create new SAS data set

- Less efficient: subsetting IF statement
  - Read observation into PDV, test condition

    If true, continue processing observation

    If false, discard observation

- More efficient: WHERE statement
  - Test condition before observation read into PDV

    If true, read observation into PDV, process it

    If false, don't read observation

Method 1, less efficient

```
data two;
  set one;
  if sales ne 0
     and income > 10;
  more SAS statements
run;
```

Method 2, more efficient

```
data two;
  set one;
    where sales ne 0
       and income > 10;
  more SAS statements
run;
```

2. Subset SAS data set, create new SAS data set

- Exception: SAS functions
- WHERE cannot use
  - Variables created in current DATA step
  - Automatic variables: FIRST., LAST., _N_, etc.
- Caveat: might not be very different any more

3. Subset SAS data set to use in PROC step

- Less efficient #1
  - Subsetting IF statement, create unnecessary data set
- Less efficient #2
  - WHERE statement, but create unnecessary data set
- More efficient
  - Need data set only for subsequent PROC step
  - WHERE statement in PROC step, eliminate DATA step

Method 1, less efficient

```
data two;
  set one;
  if sales > 0;
run;
proc print data = two;
run;
```

Method 2, less efficient

```
data two;
  set one;
    where sales > 0;
run;
proc print data = two;
run;
```

Method 3, more efficient

```
proc print data = one;
  where sales > 0;
run;
```

4. Test multiple conditions with IN, OR, or AND operators

- Boolean short circuiting: SAS tests minimum number of conditions needed to decide if expression true or false
- OR operator with multiple conditions

```
if sales > 900 or income > 500 ;
if ctry = 'czech' or ctry = 'hungary' ;
```

- IN operator with multiple conditions

```
if ctry in ('czech', 'hungary') ;
```

- AND operator with multiple conditions

```
if sales > 900 and income > 500 ;
```

4. Test multiple conditions with IN, OR, or AND operators

- Order conditions to take advantage of boolean short circuiting
- OR operators and IN operator: order from most likely to least likely
- AND operators: order from least likely to most likely
- How to determine ordering
  - Prior knowledge of data
  - PROC FREQ
- IN operator versus OR operators
  - IN operator slightly more efficient, can notice if many conditions
  - IN operator easier to read

Example 1: order to find true value ASAP

- 50,000 observations, CNTY expected to be
  - ‘czech’ in ~20,000 observations
  - ‘hungary’ in ~5,000 observations
  - ‘belgium’ in ~2000 observations
  - ‘slovakia’ in ~1000 observations

```
data two;
  set one;
  if ctry in (‘czech’, ‘hungary’, ‘belgium’, ’slovakia’);

data two;
  set one;
  if ctry =‘czech’ or ctry=‘hungary’ or ctry=‘belgium’ or
   ctry=’slovakia’;
```

Example 2: order to find true value ASAP

- 50,000 observations, CNTY expected to be
  - ‘czech’ in ~20,000 observations
  - ‘hungary’ in ~5,000 observations
  - ‘belgium’ in ~2000 observations
  - ‘slovakia’ in ~1000 observations

```
data two;
  set one;
  where ctry in (‘czech’, ’slovakia’);

data two;
  set one;
  where ctry =‘czech’ or ctry=’slovakia’;
```

Example 3: order to find false value ASAP

- 50,000 observations,
  - CNTY expected to be ‘czech’ in ~20,000 observations
  - SALES expected to be > 900 in ~10,000 observations
  - INCOME expected to be > 500 in ~1000 observations

```
data two;
  set one;
  where income > 500 and sales > 900 and
    ctry='czech';
```

SAS sometimes converts series of ORs to IN operator to improve efficiency

- No known rule for when
- Example in SAS 9.4TS1M4 Windows, SAS 9.4TS1M6 Linux

```
data test;
   myvalue=1;   run;
data test2;
  set test;
  where myvalue=1 or myvalue=2 or myvalue=3 or myvalue=4;
run;

4         data test2;
5           set test;
6           where myvalue=1 or myvalue=2 or myvalue=3 or myvalue=4;
7         run;
NOTE: There were 1 observations read from the data set WORK.TEST.
      WHERE myvalue in (1, 2, 3, 4);
NOTE: The data set WORK.TEST2 has 1 observations and 1 variables.
```

5. Subset SAS data set w/WHERE statement, use WHERE statement operators

- BETWEEN-AND operator
- CONTAINS or ? operator
- IS NULL or IS MISSING operator
- LIKE operator
- SOUNDS LIKE (=*) operator
- SAME-AND operator

WHERE statement operators: BETWEEN-AND operator

- Is variable value in a range
- Convenient syntax, no new functionality
- Equivalent statements:

```
where salary between 40 and 50;
where salary >=40 and salary <= 50;
where 40 <=salary <= 50;
```

WHERE statement operators: CONTAINS or ? operator

- Tests if character variable contains a string
- Equivalent: CONTAINS, ?
- Comparable to INDEX, INDEXC functions
- Can use instead of INDEX, INDEXC in WHERE
- Example: observations in which value of NAME=
  - JONES, JOHN, HOJO Selected
  - JAO, John Not selected

```
data two;
  set one;
  where name contains 'JO';
```

WHERE statement operators: IS NULL or IS MISSING operator

- Tests if a variable is missing
- Equivalent: IS NULL, IS MISSING
- Avoid error if not sure if variable is numeric or character
- Less self-documenting than . or ""

```
For numeric variable

where sales is null;
where sales is missing;
where sales = .;
```

```
For character variable

where name is not null;
where name is not missing;
where name ne “”;
```

WHERE statement operators: LIKE operator

- Tests if character variable contains a pattern
- Uses two wild cards
  - % matches any number of characters (0 or more)
  - _ matches any single character
- Some features of UNIX command grep
- Much more powerful than INDEX, INDEXC functions
- Much less powerful than regular expressions, simpler

1. where name like ‘J%n’;

starts with ‘J’
then 0 or more characters (any characters)
ends with ‘n’

Selected: John, Jan, Jn, Johanson
Not selected: Jonas JAN, AJohn

2. where name like ‘J_n’;

starts with ‘J’
then any single character
ends with ‘n’

Selected: Jan
Not selected: John, Jonas, Jn, Johanson, JAN, AJohn

3. where name like '%J%';

contains a 'J'

Selected: Jan, John, Jn, Johanson, Jonas, JAN, AJohn, TAJ, J

Not selected: j, Elijah, jan

WHERE statement operators: SOUNDS LIKE (=*) operator

- Uses Soundex algorithm to test character variable for spelling variation of a word
- Useful for ???

```
data one;
  input tankname $1-20;
  datalines;
new york tank
new york tank.
NEW YORK tank
new york tnk
new yrk tank
ne york tank
neww york tank
nnew york tank
ew york tank
new york mets
;run;
```

Matches all but last 2:

```
data two;
  set one;
  where tankname =*
     'new york tank';
```

Leading characters missing

```
where tankname =* 'new york tank'
or tankname =* 'ew york tank'
or tankname =* 'w york tank'
or tankname =* 'york tank';
```

WHERE statement operators: SAME-AND operator

- Adds clauses to existing WHERE statement within a DATA step
- Interactively: save keystrokes
- Makes programs harder to understand
- Not recommended

6. Read SAS data set, need only few variables in DATA step or output data set

- Less efficient #1
  - Read all variables into PDV
  - Write all variables to data set
- Less efficient #2
  - Read all variables into PDV
  - Write only needed variables to data set w/KEEP or DROP statement
- More efficient
  - Use KEEP= or DROP= to read and write only needed variables

ONE: variables A, X1-X1000

TWO: variables A, B

Method 1: less efficient

```
data two;
  set one;
  b = a * 1000;
run;
```

Method 3: incorrect result

```
data two (keep=b);
  set one (keep=a);
  b = a * 1000;
run;
```

Method 2: less efficient

```
data two;
  set one;
  keep a b;
  b = a * 1000;
run;
```

Method 4: more efficient

```
data two;
  set one (keep=a);
  b = a * 1000;
run;
```

7. Concatenate one SAS data set to another

- Assume variables in both data sets: same length, type
- Less efficient
  - SET statement, reads both data sets

```
data one;
  set one two;
```

- More efficient
  - PROC APPEND, reads only second data set

```
proc append base = one data = two;
```

  - Make BASE= the larger data set if order does not matter

8. DATA step, don't need output data set

- When? PUT statements, CALL SYMPUT
- Less efficient #1: create unneeded data set two

```
data two;
  set one;
  put @1 var1 @11 var2;
```

- Less efficient #2: recreate data set one

```
data one;
  set one;
  put @1 var1 @11 var2;
```

- More efficient: data _null_, no data set created

```
data _null_;
  set one;
  put @1 var1 @11 var2;
```

9. In DATA step, denominator of division could be zero

- Method 1, less efficient: always divide - if denominator is 0, SAS tries to divide by 0, sets quotient to missing

```
data two;
  set one;
  quotient = num/denom;
```

- Method 2, more efficient: test denominator, only divide when denominator is not 0

```
data two;
  set one;
  if denom ne 0
    then quotient = num/denom;
    else quotient = .;
```

- Method 3, more efficient but different result

```
data two;
  set one;
  quotient = divide(num,denom);
```

- When denominator=0:
  - Method 2 sets quotient to .
  - Method 3 sets quotient as follows:

Value of NUM	Value of QUOTIENT
Positive value	.I
Negative value	.M
0 or .	.
Any special numeric missing value	Value of NUM (e.g, if NUM=.M, QUOTIENT=.M)

Avoid divide by zero in PROC SQL too

- Method 1, less efficient: always divide

```
proc sql;
create table two as
    select num/denom as quotient
    from one;
quit;
```

- Method 2, more efficient

```
proc sql;
create table two as
    select
    case
      when denom =  0 then . else num/denom
    end as quotient
    from one;    quit;
```

Recent divide by zero test

- Linux SAS 9.4TS1M6
- 1 million observations, 50 numeric variables (i.e., 50 million values)
- Divide as follows for all values: 10/*value*
- 50 million divides, 20% (10 million) w/denominator=0
- Method 1: 34 seconds
- Method 2 or 3: 2 seconds

For more information, please contact

Bruce Gilsen
Federal Reserve Board, mail stop N-122
Washington, DC 20551
phone: 202-452-2494
e-mail: bruce.gilsen@frb.gov