Exploring and Validating Data

Dr. Lendie Follett

2022-04-06

Exploring Data

Proc print

- We've already used proc print to see the data portion of a SAS data set.
- By default, this will print each and every row and column of the requested SAS data set.

```
proc print data = input-table(obs = n);
run;
```

- Replace `input-table` with name (and library) of SAS data set.
- Replace `n` with the number of rows you want to print. (Helpful with large datasets.)

Your Turn

The goal of this Your Turn is to begin using SAS documentation, which will be an important part of your daily SAS use in practice.

- Go to support.sas.com/documentation (*wouldn't hurt to bookmark this*).
- Navigate to "SAS Programming: SAS 9.4 and Viya 3.5".
- Under Syntax Quick Links -> Quick Links for SAS Procedures, click "SAS Procedures by Name"
- Find the PRINT procedure in the alphabetized list of procedures.
- Examine the syntax and the table of statements that can be used with proc print.
- What is the statement that allows you to select certain variables to print? Can you choose the order they appear?

Proc print

```
/*print whole SAS data set*/
proc print data=sashelp.cars;
run;

/*print first 10 rows of these variables in specified order*/
proc print data=sashelp.cars (obs=10);
    var Make Model Type MSRP;
run;
```

Obs	Make	Model	Type	MSRP
1	Acura	MDX	SUV	$36,945
2	Acura	RSX Type S 2dr	Sedan	$23,820
3	Acura	TSX 4dr	Sedan	$26,990
4	Acura	TL 4dr	Sedan	$33,195
5	Acura	3.5 RL 4dr	Sedan	$43,755
6	Acura	3.5 RL w/Navigation 4dr	Sedan	$46,100
7	Acura	NSX coupe 2dr manual S	Sports	$89,765
8	Audi	A4 1.8T 4dr	Sedan	$25,940
9	Audi	A41.8T convertible 2dr	Sedan	$35,940
10	Audi	A4 3.0 4dr	Sedan	$31,840

Proc means

- You'll often be required to generate summary statistics of numeric columns

```
proc means data()=input-table;
        var col-name(s);
run;
```

- The var statement isn't required; omitting it will result in all *numeric* variables being summarised.
- Replace `col-name(s)` with names of columns you want summary statistics for.

```
proc means data=sashelp.cars;
    var EngineSize Horsepower MPG_City MPG_Highway;
run;
```

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
EngineSize	Engine Size (L)	428	3.1967290	1.1085947	1.3000000	8.3000000
Horsepower		428	215.8855140	71.8360316	73.0000000	500.0000000
MPG_City	MPG (City)	428	20.0607477	5.2382176	10.0000000	60.0000000
MPG_Highway	MPG (Highway)	428	26.8434579	5.7412007	12.0000000	66.0000000

Proc univariate

- Proc univariate is like proc means
- However, it is more detailed
- Just as with means and print, you can use a var statement to select columns to analyze

```
proc univariate data=input-table;
        var col-name(s);
run;
```

For example,

```
proc univariate data=sashelp.cars;
    var MPG_Highway;
run;
```

will give all the statistics `proc means` provides, but also things like a wide range of quantiles, range, etc...

Proc freq

- `proc means` and `proc univariate` are for analyzing distribution of numeric columns
- `proc freq` provides a table that counts the number of rows/observations that each value of a variable appears, i.e., a frequency table.

```
proc freq data=input-table;
    tables col-name(s);
run;
```

- The tables statement is like the var statement we saw previously; used to limit the variables/columns that sas analyzes.

Proc freq

```
proc freq data=sashelp.cars;
    tables Origin Type DriveTrain;
run;
```

- This `proc freq` step creates a separate table for Origin, Type, and DriveTrain. For example:

The FREQ Procedure

Origin	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Asia	158	36.92	158	36.92
Europe	123	28.74	281	65.65
USA	147	34.35	428	100.00

- `proc freq` is ESSENTIAL to cleaning data; checking for values that should/shouldn't be there!

Case Study

Goals:

1. Practice using libref
2. Practice four procedures just learned in the context of how you might use them in real life.

```
/*************************************************/
/*     Procedure syntax                          */
/*************************************************/
/*  Syntax                                       */
/*                                               */
/*    PROC PRINT DATA=input-table(OBS=n);        */
/*        VAR col-name(s);                       */
/*    RUN;                                       */
/*                                               */
/*    PROC MEANS DATA=input-table;               */
/*        VAR col-name(s);                       */
/*    RUN;                                       */
/*                                               */
/*    PROC UNIVARIATE DATA=input-table;          */
/*        VAR col-name(s);                       */
/*    RUN;                                       */
/*                                               */
/*    PROC FREQ DATA=input-table;                */
/*        TABLES col-name(s);                    */
/*    RUN;                                       */
/*************************************************/
```


Filtering Rows

Filtering Rows

- We've learned how to filter proc print output to only show the first n rows.

```
/*print first 10 rows*/
proc print data = sashelp.cars(n = 10);
run;
```

- It's also possible to logically filter rows based on data criteria.
- This criteria is specified in a **where statement**.

```
PROC procedure-name . . . ; /*this works for many procs*/
       WHERE expression; /*expression is some logical expression*/
RUN;
```

- `procedure-name` can be replaced with `PRINT`, `MEANS`, `FREQ`, `UNIVARIATE`, many others!
- replace `expression` with some logical statement defining rows to keep.

Filtering Rows

- We've learned how to filter proc print output to only show the first n rows.

```
/*print first 10 rows*/
proc print data = sashelp.cars(n = 10);
run;
```

- It's also possible to logically filter rows based on data criteria.
- This criteria is specified in a **where statement**.

```
/*print rows from cars data where the type is*/
proc print data = sashelp.cars;
where type = "SUV";
run;
```

Obs	Make	Model	Type	Origin	DriveTrain	MSRP	Invoice
1	Acura	MDX	SUV	Asia	All	$36,945	$33,337
27	BMW	X3 3.0i	SUV	Europe	All	$37,000	$33,873
28	BMW	X5 4.4i	SUV	Europe	All	$52,195	$47,720
47	Buick	Rainier	SUV	USA	All	$37,895	$34,357
48	Buick	Rendezvous CX	SUV	USA	Front	$26,545	$24,085
56	Cadillac	Escalade	SUV	USA	Front	$52,795	$48,377
57	Cadillac	SRX V8	SUV	USA	Front	$46,995	$43,523
64	Chevrolet	Suburban 1500 LT	SUV	USA	Front	$42,735	$37,422

Filtering Rows

Logical Expressions

Comparison operators perform a comparison, usually between a column and a constant. Expressions involving comparison operators are called **logical expressions** and evaluate to 0 (FALSE) or 1 (TRUE).

Definition	Symbol	Task.Example	Code
equal	=	does x equal y	x = y
less than	<	is x less than y	x < y
greater than	>	is x greater than y	x > y
less than or equal	`≤`	is x less than or equal to y	x <= y
greater than or equal	`≥`	is x greater than or equal to y	x >= y
not equal	`≠`	is x not equal to y	x ~ y OR x ^=y

Filtering Rows

Logical Expressions

You can use logical expressions involving character and numeric columns. For example,

```
where type = "SUV";
where MPG > 35;
where MSRP <= 30000;
```

- Character values are case sensitive and must be enclosed in double or single quotation marks
- Numeric values must be standard numeric (that is, no symbols).

Filtering Rows

Logical Expressions

What about date columns? We can involve date columns in logical expressions by using **SAS date constants**.

```
proc procedure-name . . .;
where column-name comparison-operator "ddmmmyyyy"d;
run;
```

For example,

```
where date > "01JAN2015"d;

/*OR*/

where date > "1jan15"d;
```

would subset to the rows where the date occured after January 1st, 2015.

Filtering Rows

Combining Expressions using Logical Operators

- You can combine sub-logical expressions to create a more complicated logical expression.
- Combine them using "AND", "OR", or "NOT"

Definition	Code.Symbol	Code.Mnemonic
and	&	AND
or	|	OR
not	~	NOT

For example,

```
proc print data=sashelp.cars;
    var Make Model Type MSRP MPG_City MPG_Highway;
    where Type="SUV" and MSRP <= 30000; /*both need to evaluate to TRUE to print r
run;
```


Filtering Rows

The in operator

- The **in operator** can be used to check if a value (typically of a column) is contained within a specified list

Example.task	Code
x equals 1,2,3,4,5,6,7,8,9, or 10	x in (1,2,3,4,5,6,7,8,9,10)
x equals 1,2,3,4,5,6,7,8,9, or 10	x in (1:10)
x equals 1,2, 3 or 7,8,9	x in (1:3, 7:9)
type equals SUV or sedan	type in ("SUV","Sedan")

- note: for character comparisons, the values in the list are enclosed in " "
- note: elements are enclosed in () and separated by a comma
- The in operator can make your code simpler:

```
/*these two rows of code will evaluate identically*/

where Type in ("SUV","Truck","Wagon");
```


Filtering Rows

More on the where statement

- Sometimes you want to investigate the presence of missing values
- There is special syntax for that:

```
WHERE col-name IS MISSING;
WHERE col-name IS NOT MISSING;
```

It's good to remember that SAS treats numeric and character missing values differenty. However, the code above works for both types. For example,

```
/*only look at rows with missing age*/
where age is missing;

/*only look at rows where the name is NOT missing*/
where name is not missing;
```


Filtering Rows

More on the where statement

- The **like operator** can be useful when filtering based on text/character data.

```
where col-name like "value";
```

- This allows us to do patterm matching. Usually it is uses with "%", which is a wildcard for any number of characters.
- Note: without any wildcard symbols, `like` behaves just like `=`.

Filtering Rows

More on the where statement

City
Ames
West Des Moines
Des Moines
West Des Moines
Waukee
Ankeny
Urbandale

- For example, what if I wanted to apply a filter that identified cities *ending* with "Des Moines"?

```
where City like "%Des Moines";
```


Filtering Rows

More on the where statement

City
Ames
West Des Moines
Des Moines
West Des Moines
Waukee
Ankeny
Urbandale

- Or, for some reason, that identified cities *starting* with the letter "A"?

```
where City like "A%";
```

- Bonus: the underscore "_" is a wildcard for a *single* character

Your Turn

Continue with your script t3_cs_proc_intro.sas that we started together earlier.

1. Type this code into your SAS editor.

```
proc print data=work.ss;
    *where MinPressure is missing; /*same as MinPressure = .*/
    *where Type is not missing; /*same as Type ne " "*/
    *where MaxWindMPH between 150 and 155;
    *where Basin like "_I";
run;
```

1. Uncomment each WHERE statement one at a time and run the step to observe the rows that are included in the results.
2. Comment all previous WHERE statements. Add a new WHERE statement to print storms that begin with Z. How many storms are included in the results?

Efficiency in SAS

Efficiency in SAS

Specifically, using Macro variables

Let's say you wrote your whole program around an interest in "Wagon" type vehicles. You thought your boss was really into wagons.

```
proc print data=sashelp.cars;
    where Type="Wagon";
    var Type Make Model MSRP;
run;

proc means data=sashelp.cars;
    where Type="Wagon";
    var MSRP MPG_Highway;
run;

proc freq data=sashelp.cars;
    where Type="Wagon";
tables Origin Make;
run;
```

"*Ope, actually I wanted "SUV" descriptions, not "Wagon" descriptions*".

.....

Macro variables

There is an easy way to make your programs flexible for filtering... and beyond! The solution is to create a **macro variable**.

```
/*Define a macro variable called 'macro-variable-name' with the value 'value'*/
%LET macro-variable-name = value;
```

- Replace `macro-variable-name` with whatever variable name you wish; make it meaningful but concise.
- Replace `value` with the value you want the variable to take on.
- Oddly enough, character variables don't get enclosed in " "

Use the variable by calling

```
&macro-variable-name
```

wherever you want that value used in your code.

Macro variables

What you code:

```
%let CarType = Wagon;

proc print data=sashelp.cars;
    where Type="&CarType";
    var Type Make Model MSRP;
run;

proc means data=sashelp.cars;
    where Type="&CarType";
    var MSRP MPG_Highway;
run;

proc freq data=sashelp.cars;
    where Type="&CarType";
tables Origin Make;
run;
```

What SAS reads:

```
%let CarType = Wagon;

proc print data=sashelp.cars;
    where Type=Wagon;
    var Type Make Model MSRP;
run;

proc means data=sashelp.cars;
    where Type=Wagon;
    var MSRP MPG_Highway;
run;

proc freq data=sashelp.cars;
    where Type=Wagon;
tables Origin Make;
run;
```

Everywhere SAS sees `&CarType`, Wagon replaces it!

Case Study

Using Storm Summary Data

Goals: In each of the procs we use, let's write the code to efficiently filter, in case we change our minds about the basin we're interested in.

1. Obtain simple summary statistics of MaxWindMPH and MinPressure, but only for Basin "NA".
2. Obtain a frequency table of Type, but only for Basin "NA".

Save this script as t3_cs_macrovar.sas

Formatting Columns

Formatting Columns

- Sometimes when you're exploring data, it can be difficult to interpret the raw values in the data.
- For example, SAS stores date values as the number of days after (+) or before (-) January 01, 1960.


```
proc print data=input-table;
   format col-name(s)  format;
run;
```

- `format` will take on the form of `<$>format-name<w>.<d>`
  - A `$` indicates this variable should have a character format
  - `w` represents the total width of the formatted value (if applicable)
  - All formats will include a `.`
  - The d represents the number of decimal places (if applicable)

Examples of Numeric Column Formats

Format Name	Example Value	Format Applied	Formatted Value
w.d	31415.93	5.	31415
w.d	31415.93	8.1	31415.9
commaw.d	31415.93	comma8.1	31,415.9
dollarw.d	31415.93	dollar10.2	$31,415.93
dollarw.d	31415.93	dollar10.	$31,415

Your Turn

1. Go to support.sas.com/documentation. Click Programming: SAS 9.4 and Viya.
2. In the Syntax - Quick Links section, under Language Elements, select Formats.
3. What does the datew.d format do?

In each of the following, consider how would June 17, 2021 be displayed if formatted:

1. using format date7. vs format date 9.?
2. using mmddyy10. format do? The monyy7. format?
3. using the monname. format do?
4. using the weekdate format do?

Take notes of these for easy reference!

Case Study

GOAL: Using the `class_birthdate` data, print the data for all students older than 13. Now, and going forward, be sure to create reports in a way that will make the most sense to a general audience.

Save this script as t3_cs_formats.sas.

The Sort Step

```
proc sort data=input-table <OUT=output-table>;
         BY <DESCENDING> col-name(s);
RUN;
```

- Code within `< >` is optional.
- Omitting the `out =` argument will overwrite the existing data set with the sorted output.
- Including `<descending>` will sort in descending order rather than ascending.
- You can sort by multiple columns!

```
proc sort data = s40dat.class_birthdate out = class_birthdate_sorted;
  by Sex descending Birthdate;
run;

/*
As an aside: we specified out = class_birthdate_sorted.
This implicitly saved the data set class_birthdate_sorted in the temporary work li
we could have EQUIVALENTLY specified out = work.class_birthdate_sorted.*/
```