

Robust Outlier Identification using SAS®

Jack Shoemaker, North Haven, CT

Raw data from transaction systems contain all manner of noise and pollution which can confound analysis and reporting unless identified and removed. This paper, originally present at BASUG, presents an applied statistical technique to identify outliers in raw data using tools available in Base SAS.

Comparing Data Using Measures of Location

A common task for a data analyst is comparing two or more sets of data to each other, or to an established reference value. For example, imagine an automated software product that will execute a simple Oracle transaction as if it were an Oracle user sitting at a terminal. The product records the time it takes for the system to respond to the transaction, thus measuring the response time faced by a typical user. This process is repeated twelve times a day, five times a week, for four weeks. At the end of this four week period, we have a sample of 240 (=12 x 5 x 4) observations of response time for each transaction executed at a client site. Part of the service we provide to our clients is comparing their response times to the response times of other clients in our data base.

To carry out this type of comparison, it is helpful to reduce the sample of data for each client to a single value representing the whole sample, and then compare these values across clients. The formal name for this class of reference values is measure of location. The mean, mode, and median of a data distribution are all examples of measures of location of those data. For a variety of reasons, both technical and historical, we use the mean as the preferred measure of location when comparing data distributions across samples.

The Problem of Extreme Values

Unfortunately, the mean is the least robust of the three measures of location listed above. That is, the mean is influenced by data points lying farthest away from itself. To make matters worse, the farther away these extremes become, the greater their influence on the mean. Take away one or two of these extreme values and the mean changes significantly. On the other hand, take away those same extreme values and the mode and median change only slightly, if at all. This point is illustrated by the following data:

OBS	Y90	Y80	N Obs	Variable	N	Mean
1	90.80	88.95				
2	89.41	90.84	10	Y80	10	95.64
3	76.05	100.11		Y90	10	85.73
4	90.99	62.08				
5	91.15	86.33				
6	87.71	79.86				
7	85.79	60.10				
8	80.74	83.40				
9	73.02	104.73				
10	91.60	200.00				

The variable Y90 was drawn from a normally distributed random sample with mean 90 and variance 10. The variable Y80 was drawn from a normally distributed

random sample with mean 80 and variance 10, except for observation 10. In this example we know the true population parameters for each of the variables, so, we expect our comparison to tell us that Y90 is bigger than Y80. Unfortunately, a comparison of the means of these two variables yields exactly the opposite conclusion. If observation 10 is treated as an extreme value for Y80 and set to missing for the purposes of calculating a mean, then the re-figured mean for Y80 becomes 84.04. Our comparison of means now places Y80 and Y90 in their proper order. The process of changing values to missing is known as trimming the data.

Identifying Outliers

A common rule-of-thumb taught in elementary statistics is that 95% of the data in a distribution which is normally distributed will lie within 1.96 standard deviates of the mean of the distribution. Therefore, you can treat any values more extreme than the mean \bar{n} 1.96 standard deviates as outliers and discard them from analysis. Unfortunately, like its mathematical cousin the mean, the sample variance of a distribution, and therefore its standard deviation, also suffers from the heavy influence of extreme values. So, using the rule-of-thumb listed above doesn't always yield the desired result because the standard deviation is being made too large by the very values that you would otherwise like to discard from the analysis. In the mid-seventies, a more robust technique was proposed by Tukey. His method: treat any value greater than the 75th percentile plus 1.5 times the inter-quartile distance, or less than the 25th percentile minus 1.5 times the inter-quartile distance as an outlier. This technique is robust because it uses the quartile values instead of variance to describe the spread of the data. And, quartiles are less influenced by extreme values. For a frame of reference, let's calculate the corresponding alpha value produced if the robust technique rule were applied to the gold standard of all data analysis, a standard normal distribution (mean 0, variance 1), . The data step below uses SAS to calculate this value:

```

1  data _null_;
2  q1 = probit(.25);
3  q3 = probit(.75);
4  iqr = q3 - q1;
5  lb = q1 - (1.5 * iqr);
6  ub = q3 + (1.5 * iqr);
7  tail1 = probnorm(lb);
8  tail3 = 1 - probnorm(ub);
9  alpha = tail1 + tail3;
10 put / 'Results of Robust Outlier Identification'
11 / 'On A Standard Normal Distribution:' /
12 / +5 'Z-Value of 25th Percentile.....' q1
13 / +5 'Z-Value of 75th Percentile.....' q3
14 / +5 'Inter-Quartile Range.....' iqr
15 / +5 'Lower Bound.....' lb
16 / +5 'Upper Bound.....' ub /
17 / +5 'Area Under Lower Tail.....' tail1
18 / +5 'Area Under Upper Tail.....' tail3
19 / +5 'Two-Sided Alpha Value.....' alpha;

```

The SAS log displays these results:

```

Results of Robust Outlier Identification
On A Standard Normal Distribution:

```

Z-Value of 25th Percentile.....	-0.67449
Z-Value of 75th Percentile.....	0.67449
Inter-Quartile Range.....	1.34898
Lower Bound.....	-2.69796
Upper Bound.....	2.69796
Area Under Lower Tail.....	0.00349
Area Under Upper Tail.....	0.00349
Two-Sided Alpha Value.....	0.00698

This is also the algorithm that the UNIVARIATE procedure uses to construct box plots when the plot option is specified. For example, Figure 1 at the end of this paper shows the stem-and-leaf and box plot produced for a distribution of 10,000 random values from a standard normal distribution.

Looking at the box plot on the far right of the display, the inter-quartile distance is the distance between the top and bottom of the dashed box; the whiskers, or vertical bars, extend 1.5 inter-quartile distances above and below the box. Values more extreme than this are shown as 0. (As an extension to this algorithm, values which are more than 3 inter-quartile distances above or below the box are marked with an asterisk).

Trimming Data In The Real World

Returning to our example, assume that in addition to the transaction response time described above, the monitoring product also collects a variety of other computer performance and resource utilization data associated with each execution of the benchmark transactions. The distributions of these raw data often suffer from skewness, or asymmetry, and kurtosis, or heavy tails. These defects would cause the raw means of our data samples to overstate what we believe to be the proper measures of location.

Requirements

To ameliorate the problems described above, we wish to subject all the data we gathered to the trimming algorithm before we calculated sample means and performed our other comparative analyses. Furthermore, we wanted to pass through the data set once treating all the analytic variables at the same time. Finally, we wanted to retain the raw data values even if we decided to drop them from the mean calculation. To satisfy these requirements, we developed the macros presented in this paper. The first macro, **%IDOut()**, accepts a raw input data set and creates a new data set containing a one-character trimming flag variable for each analytic variable in the raw data set. This macro also creates a new data set containing only those observations with the trimming flag set. The macro called **%OUTRep()** produces an outlier report using this data set. Finally, the macro called **%TrimRep()** uses the raw data set and the data set containing the trimming flags to create a table of trimmed means for the analytic variables.

Sample Data Set

The basic data architecture assumed by these macros is that the raw data is made up of classification, identification, and analysis variables. The classification variables define the level, or levels, at which the trimming will take place; the identification variables contain other

descriptive information which we want to retain; and, the analytic variables contain the measures we are interested in analyzing. To demonstrate the use and application of the macro system, I have created a sample data set using the following data step:

```
%SetUp
data sasave.raw;
format _numeric_ 7.4;
do x = 1 to 1000;
  y = mod(x,3);
  std_norm = rannor(32983);
  evenly = ranuni(58941);
  longtail = ranpoi(23457,1) + evenly - 1;
output;
end;
run;
```

Our data set has three analytic variables - `std_norm`, `evenly`, and `longtail`; one classification variable, `y`; and, one identification variable, `x`. Pages 1 through 3 of the handout show the PROC UNIVARIATE output for the three analytic variables created in the data step above. As you will see, each of the macros make use of lists of variable names, labels, and titles. These lists are stored in the four short macros shown in Figure 2. The macro called **%RawVars** contains a list of these three analytic variables. To implement the trimming algorithm, we need to calculate a variety of rank statistics for each variable in the **%RawVars** list. This means we need a new variable name for each statistic for each analytic variable. For example, if we need seven rank statistics for three analytic variables, then we need twenty one new variable names. To control and standardize the variable naming conventions, we decided that each new variable would have the following form: `VN_TAG`, where `VN` is a two-character abbreviation for the raw variable name, and `TAG` is a pre-defined suffix corresponding to each of the rank statistics. For example, the 'TAG' value for median is 'md'. The macro called **%TheVars()** accepts `__TAG` as a keyword parameter and generates a list of new variable names following our standard naming convention. Note that the order of variable names in **%TheVars()** corresponds to the order in **%RawVars**. The other two macros in Figure 2 contain two ^-delimited lists of full variable names and variable labels used by the reporting macros.

The balance of this paper is devoted to describing the use and execution of the automatic trimming application made up of **%IDOut()**, **%OutRep()**, and **%TrimRep()**.

%IDOut()

The full source for **%IDOut()** is found in Figure 3. A call to **%IDOut()** using our sample data produces the log found on the next page:

```

90   %IDOut(__INDSN=sasave.raw,
91         __FLGDSN=sasave.flags,
92         __OUTDSN=sasave.outlier,
93         __LVL=y,
94         __ID=x,
95         __MAC=TheVars,
96         __RAW=%RawVars);
NOTE: The data set WORK.ALL has 1000
observations and 5 variables.
NOTE: The data set WORK.RANKSTAT has 3
observations and 22 variables.
NOTE: The data set WORK.RANKSTAT has 3
observations and 29 variables.
NOTE: The data set WORK.RANKSTAT has 3
observations and 29 variables.
NOTE: The data set SASAVE.FLAGS has 1000
observations and 5 variables.
NOTE: The data set SASAVE.OUTLIER has 26
observations and 36 variables.

```

Notice the use of the **%RawVars()** and **%TheVars()** macros in the **%IDOut()** macro call. The symbol **__MAC** expects a macro name that it will use to generate new variable names in the PROC UNIVARIATE step. Here's what happens when the symbol **__MAC** is filled with the string 'TheVars'. The character string 'TheVars' is passed into **IDOut**. Everywhere that SAS encounters the symbol **&__MAC.**, it substitutes **TheVars**. So the first line under the output statement of PROC UNIVARIATE expands to:

```
q3 = %TheVars(__TAG=q3)
```

SAS detects that the line still contains percent signs or ampersands, so it re-scans the line and looks for a macro definition called **%TheVars()** so it can continue. It finds the **%TheVars()** definition and calls it using **__TAG=q3**. On the second and final pass across this line, the statement looks like this:

```
q3 = sn_q3 lt_q3 ev_q3
```

Contrast this with the symbol **__RAW** which expects a list of variable names. We could have called **%IDOut()** with a statement like:

```
%IDOut(...,
  __RAW=std_norm longtail evenly,
  ...)
```

Instead, we used the **%RawVars()** macro to generate the list for us. As you can see, the SAS Macro facility provides ample flexibility to generate lots of code from a few macro statements. The main point to remember is that SAS will continue to re-scan and substitute as long as it encounters ampersands and percent signs, or unless you have disabled scanning through the quoting functions. Also, you can use macros to fill in keyword parameters, and macros may contain calls to other macros.

Turning our attention to the log messages, we see that the work data set **RANKSTAT** has one observation per unique combination of classification variables, in this case the three values of **Y**. The permanent data set, **FLAGS**, has exactly the same number of observations as the raw data set. The five variables are the classification variable, **Y**; the

identification variable, **X**; and the flag variables **SN_T**, **LT_T**, and **EV_T**. The twenty six observations in the permanent data set **OUTLIER** are the twenty six observations that contain at least one outlier flag variable with a value of 'S'.

%OutRep()

The full source for **%OutRep()** is found in Figure 4. A call to **%OutRep()** must be made after a call to **%IDOut()**. That is, **%OutRep()** expects a data set that has already passed through **%IDOut()**. Using our sample data produces the following log:

```

61   %OUTRep(__OUTDSN=sasave.outlier,
62         __LVL=y,
63         __ID=x,
64         __RAW=%RawVars,
65         __VLST=%TheVars(),
66         __NAMLST=%VarNam,
67         __LABLST=%VarLab);
NOTE: The data set WORK.OUTLIER has 26
observations and 36 variables.
NOTE: The PROCEDURE SORT used 2.00 seconds.
NOTE: The PROCEDURE PRINT used 3.00
seconds.
NOTE: The PROCEDURE PRINT used 2.00
seconds.
WARNING: No observations were selected from
data set WORK.OUTLIER.
NOTE: The PROCEDURE PRINT used 2.00
seconds.

```

The macro executes a PROC PRINT for each variable in the variable abbreviation list, **__VLST**. Notice that when **%TheVars()** is called with a null value for **__TAG**, a simple list of abbreviations is generated. The macro function **%SCAN** is used to break the list into blank-delimited tokens. For the full name and label lists, blank delimited tokens would produce undesirable results because blanks often appear in titles and labels. We decided to delimit these strings using the **^** character instead. Obviously, if our full names or labels contained a **^**, we would select a different delimiter.

After two PROC PRINT executions, the log issues a warning stating that no observations were selected from **WORK.OUTLIER**. This corresponds to the last iteration of the loop for the raw variable **evenly**. Apparently, our algorithm did not identify any outliers for the variable **evenly**. This is expected since the variable was drawn from a uniform distribution.

%TrimRep()

The full source for **%TrimRep()** is found in Figure 5. It uses the same looping strategy as **%OutRep()** to perform a PROC SUMMARY and a PROC TABULATE for each variable in the variable abbreviation list. Using our sample data produces the following log:

```

64   proc sort data = sasave.raw;by x;
65   proc sort data=sasave.flags;by x;
NOTE: The data set SASAVE.RAW has 1000
observations and 5 variables.
66   run;
NOTE: The data set SASAVE.FLAGS has 1000
observations and 5 variables.
67
68   data flagged;
69   merge sasave.raw
70         sasave.flags;by x;
71   run;
NOTE: The data set WORK.FLAGGED has 1000
observations and 8 variables.
72
73   %TrimRep(__INDSN=flagged,
74           __LVL=y,
75           __RAW=%RawVars,
76           __VLST=%TheVars(),
77           __NAMLST=%VarNam);
NOTE: The data set WORK.TRIMMED has 3
observations and 7 variables.
NOTE: The data set WORK.TRIMMED has 3
observations and 7 variables.
NOTE: The data set WORK.TRIMMED has 3
observations and 7 variables.
seconds.
```

Notice that before **%TrimRep()** is called, we need to merge the outlier flags in the flags data set with the raw data set. A where statement in PROC SUMMARY then selects only those observations which were not flagged. This must be done for each variable individually because an observation may be an outlier of one variable but not the other.

A variation to this report (not shown here) is to promote the flag variable to a classification variable, remove the where statement, and make the classification variables by variables instead. The result is a data set which contains raw statistics, `_type_ = 0`; trimmed statistics, `_type_ = 1` and `flag = "`; and, results for the outlier observations, `_type_ = 1` and `flag = 'S'`.

The author welcomes comments, suggestions, corrections, and amplifications.

Jack N Shoemaker
North Haven, CT 06854

203 234 2884 shoe@world.std.com

SAS® is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. Other brand and product names are registered trademarks or trademarks of their respective companies.

Figure 3: %IDOut()

```

%macro IDOut(__INDSN=,          /* Raw Data Set */
             __FLGDSN=,        /* Flags Data Set */
             __OUTDSN=,        /* Outlier Data Set */
             __LVL=,           /* Classification Variables */
             __ID=,            /* ID Variables */
             __RAW=,           /* Analysis Variables */
             __MAC=,           /* Variable Generating Macro */
             __X=1.5,          /* IQR Multiplier */
             __SIG=0.0005);    /* Difference Significance */

proc sort data = &__INDSN out = all;by &__LVL.;

proc univariate data = all noprint pctldef = 3;
by &__LVL.;
var &__RAW.;
output out = rankstat
      q3 = %&__MAC.(__TAG=q3)
      q1 = %&__MAC.(__TAG=q1)
      median = %&__MAC.(__TAG=md)
      mean = %&__MAC.(__TAG=mu)
      n = %&__MAC.(__TAG=n)
      min = %&__MAC.(__TAG=mn)
      max = %&__MAC.(__TAG=mx) ;

data rankstat(drop = i);
set rankstat;
array q3{*} %&__MAC.(__TAG=q3) ;
array q1{*} %&__MAC.(__TAG=q1) ;
array mn{*} %&__MAC.(__TAG=mn) ;
array mx{*} %&__MAC.(__TAG=mx) ;
array lb{*} %&__MAC.(__TAG=lb) ;
array ub{*} %&__MAC.(__TAG=ub) ;
do i = 1 to dim(ub);
  spread = (q3{i} - q1{i}) * &__X.;
  lb{i} = max((q1{i} - spread),mn{i});
  ub{i} = min((q3{i} + spread),mx{i});
end;

proc sort data = rankstat;by &__LVL.;

data &__FLGDSN.(keep=&__LVL &__ID. %&__MAC.(__TAG=t) )
      &__OUTDSN.(drop=i);
merge all rankstat;by &__LVL.;
array tvals{*} $ 1 %&__MAC.(__TAG=t) ;
array rvals{*} &__RAW. ;
array lb{*} %&__MAC.(__TAG=lb) ;
array ub{*} %&__MAC.(__TAG=ub) ;
do i = 1 to dim(tvals);
  if (lb{i} - rvals{i}) > &__SIG then tvals{i} = 'S';
  else if (rvals{i} - ub{i}) > &__SIG then tvals{i} = 'S';
end;
do i = 1 to dim(tvals);
  if tvals{i} = 'S' then do;
    output &__OUTDSN.;
    goto nextrec;
  end;
end;
nextrec: output &__FLGDSN.;
run;

%mend IDOut;

```

Figure 4: %OutRep()

```

%macro OutRep(__OUTDSN=,      /* Outlier Data Set */
              __LVL=,        /* Classification Variables */
              __ID=,         /* ID Variables */
              __RAW=,        /* Analysis Variables */
              __VLST=,       /* Variable Name Abbreviations */
              __NAMLST=,     /* Full Variable Name List */
              __LABLST=);    /* Variable Label List */

proc sort data = &__OUTDSN. out = outlier;by &__LVL. &__ID.;
run;

%let __C = 1;
%let __V = %scan(&__VLST,&__C,%str( ));
%do %while(&__V ^=);

    %let __VR = %scan(&__RAW,&__C,%str( ));
    %let __LAB = %scan(&__LABLST,&__C,%str(^));
    %let __NAM = %scan(&__NAMLST,&__C,%str(^));

    proc print data = outlier split = '*' uniform;
    where &__V.t = 'S';
    by &__LVL. &__V.lb &__V.ub notsorted;
    id &__LVL. &__V.lb &__V.ub;
    var &__ID. &__VR. ;
    format x y 4. &__VR &__V.lb &__V.ub 12.3;
    label x = '*X*====='
           y = '*Y*====='
           &__V.lb = '*Lower Bound *====='
           &__V.ub = '*Upper Bound *====='
           &__VR = "&__LAB.";
    title3 "The Following Observations Were OUTSIDE The Allowable Range";
    title4 "For The Analysis Variable ==> &__NAM.";
    run;

    %let __C = %eval(&__C + 1);
    %let __V = %scan(&__VLST,&__C,%str( ));
%end;

%mend OutRep;

```

Figure 5: %TrimRep()

```

%macro TrimRep(  __INDSN=,          /* Input Data Set With Flags*/
                __LVL=,           /* Classification Variables */
                __ID=,            /* ID Variables */
                __FMT=9.4,       /* Default Numeric Format */
                __RAW=,          /* Raw Variable List */
                __VLST=,         /* Variable Name Abbreviation List */
                __NAMLST=);      /* Full Variable Name List */

%let __C = 1;
%let __V = %scan(&__VLST,&__C,%str( ));
%do %while(&__V ^=);

    %let __VR = %scan(&__RAW,&__C,%str( ));
    %let __NAM = %scan(&__NAMLST,&__C,%str(^));

    proc summary data = &__INDSN missing nway;
    where &__V.t ^= 'S';
    class &__LVL.;
    var &__VR.;
    output out = trimmed
           n = &__V._n min = &__V._mn max = &__V._mx mean = &__V._mu;
    run;

    proc tabulate data = trimmed missing noseps format = &__FMT.;
    class &__LVL.;
    var &__V._n &__V._mu &__V._mn &__V._mx;
    label
           y = 'Y'
           &__V._n = 'N Obs'
           &__V._mn = 'Minimum Value'
           &__V._mx = 'Maximum Value'
           &__V._mu = 'Average Value';
    table y,
           sum=' ' * (&__V._n*f=5. &__V._mu &__V._mn &__V._mx) / rts = 16 condense;
    title3 "Trimmed Results for Analysis Variable ==>  &__NAM.";
    run;

    %let __C = %eval(&__C + 1);
    %let __V = %scan(&__VLST,&__C,%str( ));
%end;
%mend TrimRep;

```