Paper 323-2010

Instant KPI: From Data to Dashboard in Record Time

Elliot Inman and Michael Drutar, SAS Institute Inc., Cary, NC

ABSTRACT

Moving from data to dashboard in a single program, this paper shows how Base SAS® and SAS/STAT® software can be used with the power of PROC GKPI to go from raw data to meaningful statistical summaries of benchmarked performance. Benchmarking is a common process by which organizations measure performance against group norms, past performance, or expected performance. For example, a school might compare its graduation rate against its school district's rate, a hospital might compare care for an individual patient with patients with similar medical histories, or a retail store might compare itself to other franchises. Using PROC MEANS, PROC UNIVARIATE, and other summaries, you can calculate statistical boundaries on the fly, renorming data using group distributions, and publishing the results to the dashboard. This paper demonstrates how stored statistical procedures can be run automatically to generate valid comparison groups and report the resulting metrics, with appropriate statistical boundaries, using Output Delivery System (ODS) graphics ready for the SAS® Information Delivery Portal, any generic HTML dashboard, or offline reporting. Applications in several industries are discussed.

INTRODUCTION

While it might seem obvious, it is worth repeating: Data have meaning only in context. Both statisticians and decision makers understand this fact, although they usually describe the situation in very different ways. Statisticians understand that a mean is meaningful only in the context of a distribution. Depending on the normality of the distribution, the skew, the kurtosis, and other aspects of how the sample was drawn from a population, the mean might be more or less representative of the distribution. Decision makers are often asking the same question, but using different words to ask it. Decision makers might ask whether a number can be trusted, whether it is based on the right fundamentals, whether it reflects what is really happening with their business, and whether the number is good or bad. Both the statistician and the decision maker are actually asking the same questions. Both will rely on analytics to provide an answer.

The business process by which companies evaluate their performance is called benchmarking. A benchmark is the ruler against which performance is measured. But benchmarks are seldom fixed values that hold constant over time. Engineers and architects can rely on pi (3.14159265…) in construction and design, just as doctors can rely on 98.6 degrees Fahrenheit as a healthy body temperature for most people. But in many cases, especially in fields as varied as elementary school education to retail coffee sales to stock portfolio management, benchmarks for "good enough" are not universally known or available.

There might be a good reason why fixed standards have not been set:

- The metric is new and no one knows what to expect in terms of the distribution of scores on the metric.
- The metric is so quickly changing that having a fixed standard would quickly become meaningless.
- Relative performance is the only important analysis of the metric.

For example, when new technologies are deployed, new measures must be developed to gauge the success of the project. At the dawn of the Internet Age, metrics for monitoring Web site use included counts of whether users had clicked through to a Web site and landed on a particular page of HTML code. Initial metrics like page views or hits were computed to measure Web site popularity. Those data were then provided to advertisers and investors interested in the success of a Web site. In the early 1990s, when popular use of the World Wide Web was growing rapidly, it was difficult to say what was good enough for the number of hits to a Web site, but you could calculate the relative frequency of hits against other Web sites or even the previous month's usage. Since then, more accurate measures have been developed including measuring rollover with a mouse and methods for distinguishing accidental versus Web bot versus dead-landing hits to distinguish a user who deliberately viewed a Web page for a length of time from false alerts of usage. At the same time, many early simple count measures are still used to determine the prevalence of branded Web browsers or search engine usages. When analysts consider the market penetration of a Web browser or search engine, they speak of the relative prevalence of engines like Google compared to Yahoo compared to Bing or the relative usage of Internet Explorer, Safari, Firefox, and other Web browsers versus each other. So, there are many cases, even in high-tech data-driven environments, where relative performance is the dominant method for measuring performance.

Many other industries take a similar approach, identifying critical metrics but relying on relative performance as a measure of success. In health care, hospitals follow accepted standards for care such as those issues by the Joint Commission, formerly the Joint Commission on Accreditation of Healthcare Organizations (JCAHO), which requires data reported on a wide range of metrics from counts of the number of healthy live births to whether patients were administered antibiotics 30 minutes before knee surgery. Numerous metrics are described in detail in the JCAHO standards. (See http://www.jointcommission.org). But hospitals also benchmark their performance against each other using a set of hospitals that are like them in terms of size, medical specialties, and patient profiles. They evaluate their performance on the metrics relative to their peers.

Colleges and universities belong to similar accrediting agencies that require reporting of metrics like the number of Ph.D.-level faculty, the number of graduates, and the financial health of the institution. (For a list of the different accrediting agencies, see http://www.chea.org.) But to determine whether graduation rates are good, bad, or good enough, colleges compare themselves to like institutions based on student population and the variety of degree programs offered. Even within a primary or secondary school for younger students, relative benchmarks can be exceptionally beneficial in understanding student performance.

Consider a high school in Raleigh, North Carolina in the United States. Imagine that the school is struggling to meet academic standards on end-of-the-year educational tests administered to all students. By law, the school must provide 180 days of instruction to all children, but of course not every child attends every day. Some students on some days will be absent due to illness or other excusable reasons. At the same time, some young people might not attend school for other, less valid reasons, skipping classes or engaging in other types of inappropriate behavior. Some students might even not be attending their regular classes as a part of a disciplinary action against them. No matter what the reason, everyone recognizes that, without being in class, students do not have the opportunity to learn. On the other hand, few students will have perfect attendance. What, then, is “good enough”? One way to answer that question is through the use of relative benchmarks.

With access to a dashboard that summarizes student performance, decision makers can answer a number of critical questions such as: What is the attendance of students in a particular classroom compared to classrooms with the best and worst attendance? If the decision maker knew only that the attendance in classroom X was about 172 days a year, that is interesting, but not very meaningful out of context. If the decision maker knew the attendance was 172 days and the highest average attendance for any classroom was 174 days and the lowest was 162 days, then 172 days assumes a different meaning. If the decision maker knew that attendance was 172 days and the average attendance among the highest scoring classrooms was 172 days and the average attendance among the lowest performing classrooms was 156 days, then that makes 172 days even more meaningful. Using data on every student in every classroom, a decision maker could have access to relative data on each classroom. This would obviously be more useful information; in fact, SAS has a software solution that does just that. (See http://www.sas.com/govedu/edu/k12/ondemand.html.)

As useful as benchmarks can be in terms of the importance of the data in context, to be really useful to decision makers, the data must be presented in an easy-to-interpret report and be updated often and delivered quickly. This paper presents an approach to meeting those goals. This approach relies on traditional SAS tools like Base SAS and SAS/STAT software and new technologies such as PROC GKPI in SAS/GRAPH® software in 9.2 for outputting easy-to-interpret dials and sliders showing performance compared in the context of a benchmark. The examples here have been kept as simple as possible to demonstrate the basic approach. At the end of the paper, we discuss a number of additional approaches that could add greater analytical depth to this reporting.

PROC GKPI: BASIC CODING

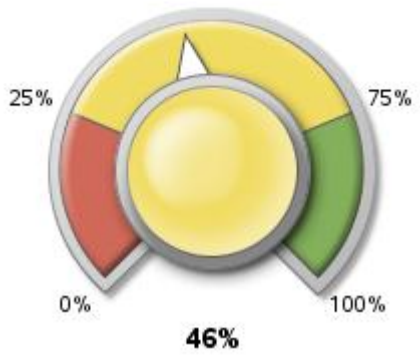
PROC GKPI is SAS code that can be used with SAS/GRAPH software in 9.2. Using the ODS graphics, PROC GKPI will produce key performance indicator (KPI) graphics like dials and sliders typically found on business intelligence dashboards. The procedure allows programmers to create a series of KPIs with only a few lines of code. The resulting KPI images output can be placed into a desktop publishing presentation as JPEG files, an executive dashboard Web site via HTML, or even exported as a PDF file. Here we give a simple example of a GKPI and discuss the various options.

The following code creates the graph shown. No additional code is required.

```
goptions reset=all device=javaimg xpixels=240 ypixels=200;
ods html style=listing;

proc gkpi mode=raised;
```

```
dial actual=.46 bounds=(0 .25 .75 1) /
   format="percent8.0"
   afont=(f="Albany AMT" height=.5cm)
   bfont=(f="Albany AMT" height=.4cm) ;
run;
quit;
ods html close;
```

The resulting output is:



There are two parts to the code. The first concerns the general setup required for SAS ODS and the output options. The second part, beginning with PROC GKPI, executes this example.

Within the setup code, there is a GOPTIONS graphics options statement. This resets all graphics options and identifies the graphic device to be the javaimg driver. Then, you must declare the size of the output graph. In this case, the horizontal plane is 240 pixels wide and the vertical plane is 200 pixels high. The **`ODS html`** command initiates ODS output and style=listing identifies an output style; many others are available. Note that at the end is **`ODS html close`**, which terminates ODS output. That is the only code wrapper required to access ODS here.

The actual KPI is generated starting with PROC GKPI. The first option declared is mode=raised. The mode option controls whether the resulting graphic will be three-dimensional or two-dimensional. The authors recommend always using the raised value for this statement as it produces a much more visually appealing graphic, more consistent with modern graphical styles. The alternative value for this statement is mode=basic, which would produce a two-dimensional graphic.

The next option referenced is “dial.” PROC GKPI has the ability to produce a wide variety of KPI images. Some of these are dials, traffic lights, horizontal bullets, and horizontal sliders. For simplicity’s sake, we will use only the dial here, but readers are encouraged to try the other options. Following the call to dial is **`actual=.46`**. This is the actual data value that the indicator is displaying. This ACTUAL statement is immediately followed by a BOUNDS= statement. The bounds of the GKPI procedure declare the ranges for the minimum and maximum data to be displayed. The bounds also declare what the ranges for green, yellow, and red will be in the resulting graph output. GKPI assumes that higher values are better, green and lower values are bad, and red is consistent with an American traffic light system with green for go and red for stop.

In our example, the minimum value for the KPI dial is **`0%`**. So the first value in the bound syntax is **`0`**. On the graph, the minimum value of 0 to 25% is red. Hence, the next value in our bounds statement is **`.25`**, which begins yellow. Other bounds are set the same way.

The next syntax in the GKPI statement is a format statement. This functions just like a format statement in any SAS DATA step. In the preceding example, the percentage format is used. This returns the values formatted 0, 25%, 75%, 100%, and 46%. Any other SAS format can be used here (such as comma8., 8., and so on). The next line of the PROC GKPI statement formats the font style and sizes for the actual text on the resulting KPI output. The “afont” option controls the format for the **`actual`** value on the indicator (in this case 46%). The “bfont” option controls the format for the tick mark values, which are around the outer rim of the dial (in this case 0, 25%, 75%, and 100%).

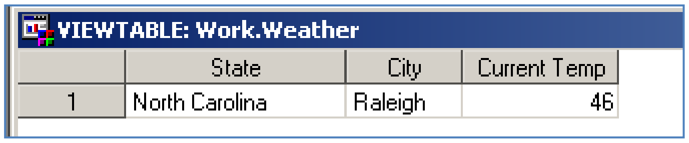
Finally, the procedure is run using a RUN statement and the ODS listing is closed. That is all of the code needed to generate a modern, attractive dial reflecting benchmarked data. The next section discusses the use of a single SAS function to populate the values within GKPI.

AUTOMATING CAPTURE OF ACTUAL VALUES

One of the unusual aspects of GKPI is that the values for the dial are set within the code. Actual and bounds data are essentially hardcoded within the PROC. The preceding section explained how the GKPI statements function; however, in order for the programmer to use this example, the programmer had to hardcode the indicator's **`actual`** value into the code. This approach is obviously not ideal, especially in the context of automatic benchmarking where we want to update the bounds or actual values often and we have a large number of disaggregated groups on which we need to report. We need to be able to pull the **`actual`** value from a SAS data set so that the indicator's **`actual`** value can change dynamically as the values in the data set change.

To facilitate dynamic data loading, we will use a single powerful function: the SYMPUT function. The SYMPUT() routine assigns the value of a DATA step variable to a macro variable. The macro variable then retains that value for the duration of the session, making the value available to multiple procedures, essentially holding the data value in memory throughout a SAS session. The programmer can then at any later point in the SAS session call this value. For example, imagine we have summarized a data table of temperatures and a data set with one row:

```
data weather;
input State $ City $ Current_Temp;
datalines;
North_Carolina Raleigh 46
;
run;
```



VIEWTABLE: Work.Weather

	State	City	Current Temp
1	North Carolina	Raleigh	46

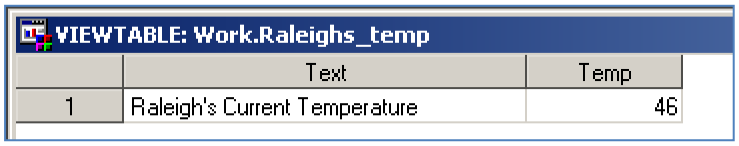
We can use the SYMPUT statement to have SAS remember the Current Temperature value of **`46`**. The code to achieve this is executed in a DATA step:

```
data _null_;
set weather;
call symput('raleigh_temp',Current_Temp);
run;
```

In the preceding code, we have told SAS to look at the data set weather and remember the value in the Current_Temp column. We can call this value later in our SAS session by using the symbolic reference call: &raleigh_temp. For example, once the preceding code has run, we can run the following code:

```
data Raleighs_Temp;
Text = "Raleigh's Current Temperature";
Temp = &raleigh_temp;
run;
```

This results in the following data set. Notice how instead of actually typing the value of **`46`** in the Temp column, we used &raleigh_temp instead. SAS has remembered that the programmer declared that the value for &raleigh_temp is **`46`** from our SYMPUT statement. The data set output would be:



VIEWTABLE: Work.Raleighs_temp

	Text	Temp
1	Raleigh's Current Temperature	46

With a method of grabbing any particular value, holding it in memory, and referencing it, we can use SYMPUT to get the values we want to use for actual and bounds data in PROC GKPI:

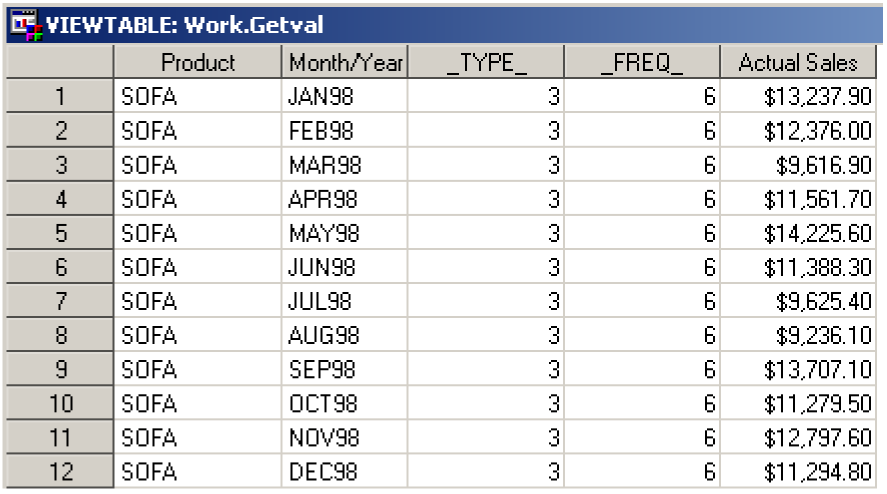
```
goptions reset=all device=javaimg xpixels=240 ypixels=200;
ods html style=listing;
proc gkpi mode=raised;
dial actual=&raleigh_temp bounds=(0 25 75 100) /
   format="8.0"
   afont=(f="Albany AMT" height=.5cm)
   bfont=(f="Albany AMT" height=.4cm) ;
run;
quit;
ods html close;
```

DYNAMICALLY SETTING BOUNDS

Up until this point, we have been using 0% to 100% as the bounds for the KPI. We could easily shift many different scales to a 0 to 100 scale if we wanted. But it is often far more helpful to report in the actual values of the metric, putting the observed data in the original context. The bounds define the benchmark. In this section, we show how to create dynamic bounds using SAS procedures to capture measures of central tendency. We use an example data set on furniture sales that is available in the SAS help library:

```
data start;
set sashelp.Prdsal2;
where year = 1998 and state = "California" and Product = "SOFA";
run;

proc means sum nway data=start noprint;
var Actual;
class Product monyr;
output out=getval sum()=;
run;
```

In this MEANS statement, we capture the product sales data for each month of the year. When we run the code, we get the resulting table:



VIEWTABLE: Work.Getval

	Product	Month/Year	_TYPE_	_FREQ_	Actual Sales
1	SOFA	JAN98	3	6	$13,237.90
2	SOFA	FEB98	3	6	$12,376.00
3	SOFA	MAR98	3	6	$9,616.90
4	SOFA	APR98	3	6	$11,561.70
5	SOFA	MAY98	3	6	$14,225.60
6	SOFA	JUN98	3	6	$11,388.30
7	SOFA	JUL98	3	6	$9,625.40
8	SOFA	AUG98	3	6	$9,236.10
9	SOFA	SEP98	3	6	$13,707.10
10	SOFA	OCT98	3	6	$11,279.50
11	SOFA	NOV98	3	6	$12,797.60
12	SOFA	DEC98	3	6	$11,294.80

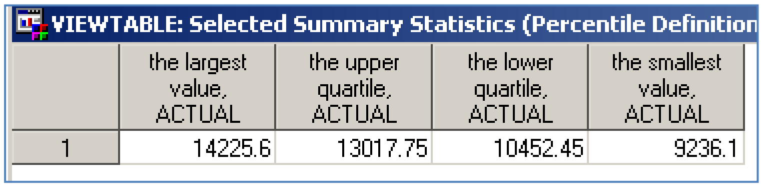
We want to use the final observation (December 1998) as our **actual** value in the KPI chart. We can use SYMPUT to capture that value:

```
data _null_;
set getval;
call symput('actual',ACTUAL);
run;
```

To get the bounds, we can use PROC UNIVARIATE to identify the minimum, maximum, quartile 1, and quartile 3 values, effectively splitting the observed distribution into four parts:

```
proc univariate data=start noprint;
var Actual;
output out=univar q1=q1 q3=q3 max=max min=min;
run;
```

The resulting data set includes the following:



VIEWTABLE: Selected Summary Statistics (Percentile Definition

	the largest value, ACTUAL	the upper quartile, ACTUAL	the lower quartile, ACTUAL	the smallest value, ACTUAL
1	14225.6	13017.75	10452.45	9236.1

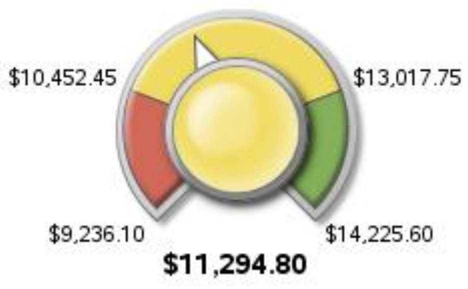
As we did with the Raleigh temperature example, we now have a data set with one row of data from which we can pull values using SYMPUT. Using the output data set univar, we can capture all of the relevant values for our KPI:

```
data  null ;
set univar;
call symput('min',min);
call symput('q1',q1);
call symput('q3',q3);
call symput('max',max);
run;
```

Now that we have these values, we will use them in our BOUNDS statement:

```
goptions reset=all device=javaimg xpixels=240 ypixels=200;
ods html style=listing;
proc gkpi mode=raised;
dial actual=&actual bounds=(&min &q1 &q3 &max) /
   format="dollar15.2"
   afont=(f="Albany AMT" height=.5cm)
   bfont=(f="Albany AMT" height=.4cm) ;
run;
quit;
ods html close;
```

The resulting output is:



This method of using PROC UNIVARIATE to generate the bounds for the KPI output allows a user to create a KPI graph from virtually any data. The ability to auto-benchmark data against itself proves valuable as it allows the programmer to give the executive dashboard view of any field. Additionally, since the indicator will dynamically change based on the underlying data, a programmer can set up an automated system where not only the actual value of the indicator is changing as the data set changes, but the bounds change, too. The programmer can take this one step further by taking the bounds from one data set and the actual value from another. That might be useful if you wanted to reference a normative benchmark that did not change as often as the new observations to be graphed such as this month's store sales against last year's monthly average. You might also compare the actual value of sofa sales with univariate bounds from a data set with all products, benchmarking the sofa product against all products in the company. That would allow decision makers to review sales of sofas in that product's own category and in the larger context of all sales.

AUTOMATICALLY RUNNING THE KPI FOR EVERY MEMBER OF A GROUP

One of the benefits of being able to dynamically fill the actual and bounds data for GKPI is that you can generate multiple KPIs for groups or individuals. For example, using the preceding sofa sales data, if this were a company with 1000 different stores, we could calculate the company's highs and lows to set the benchmarks for sales and then generate the KPI for every store in the company. Using SAS MACRO code, we could run this for every store by geographic region, by product, or by any other set of class variables. We could also do that automatically using a MACRO, updating the results as often as we choose.

To return to the example of a school interested in student attendance, imagine that we want to generate a report for each student in a school benchmarked against the student's peers. If the state has a law requiring the school to provide 180 days of instruction, that is a critical threshold. But not every student will attend every day if for no other reason than a short-term illness. So, no school could expect 180 days of attendance from every student every day. Furthermore, a significant weather or health event could radically alter the school calendar for many students. So, we might not always know what to expect. On the other hand, we do want some way of knowing if a student has missed an unusual number of days.

To do so, we need each student's data plotted on our dial, one report for each student to provide to a parent or teacher. Providing the same analysis for different students is easily accomplished with a SAS MACRO, but doing so presents another challenge. Enrollment changes from year to year. Within the year many students might leave a school and suddenly appear very absent for weeks or months if data are not carefully updated. We might even need to generate these data for a state with 100,000 students in each grade. We want to be able to produce this report for every student for whom we have currently available data without having to reference another lookup table or hardcode the macro execution using student names.

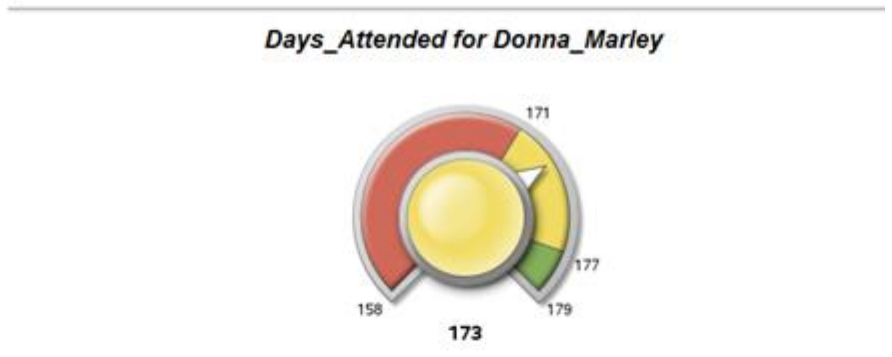
The following code includes another SAS technique that automates this reporting. By assigning each unique student a number, an integer i, and capturing the maximum value of i, we can initialize a counter for the MACRO that reports for each individual student from 1 to Max(i), without even knowing the names until we see them on the KPIs.

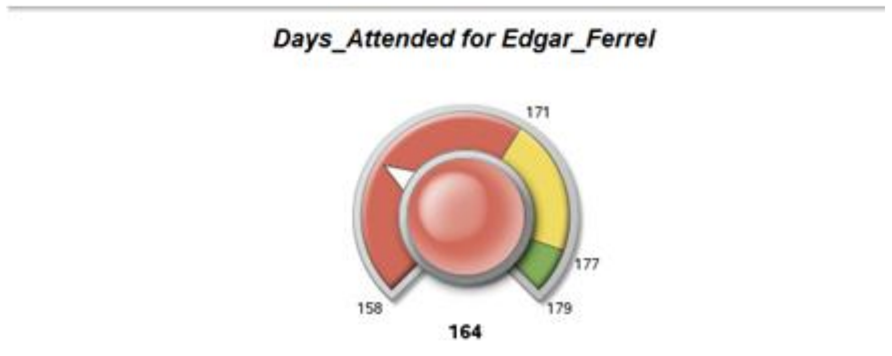
The following code creates a small data set of students, gathers the quartile levels to set the benchmarks, and plots each student on a KPI dial using the student's own data:

```
/*  WARNING:  Running this code will generate one jpeg file for each student
automatically on your computer. */

/* Simple data set of students and days attended for each */

data master;
input Student $17. Days_Attended;
datalines;
Sam Smith          178
Jake Black         172
Marcia Hanson      171
Densie Albertson   174
Rachel Long        176
Marty Neilson      177
Danny Drake        173
Edgar Ferrel       164
Susan Richards     158
Donna Marley       173
Andre_DeMint       179
;
run;
proc sort; by Student; run;

/*  Assign observation number to i; */
data master2; set master; i = _N_; run;

/* Find the last one (largest value of i) and save that to run the macro below */
proc means max data=master2; var i; output out=getmax; run;
data _NULL_; set getmax; call symput('thelast',i); where _STAT_ = "MAX"; run;
```

```
/* Get the bounds for all runs of the KPI */
proc univariate data=work.master2 noprint;
var Days Attended;
output out=univar q1=q1 q3=q3 max=max min=min;
run;

data  null ;
set univar;
call symput('min',min);
call symput('q1',q1);
call symput('q3',q3);
call symput('max',max);
run;

/*  Create a MACRO that, executed once, executes a counter within the MACRO to run
the KPI for every observation from i = 1 to the last i */

%MACRO make_print();  /* Start of the make_print MACRO */

%DO i2 = 1 %to &thelast;  /* Start of the do loop */

/* Get the actual data for this student this run */

data  null ;
set master2;
call symput('actual',Days Attended);
call symput('name',Student);
where i = &i2;
run;

ods html style=listing;
goptions reset=all device=javaimg
xpixels=240 ypixels=200;

proc gkpi mode=raised;
dial actual=&actual bounds=(&min &q1 &q3 &max) /
   format="8.0"
   afont=(f="Albany AMT" height=.5cm)
   bfont=(f="Albany AMT" height=.4cm) ;
   title "Days_Attended for &name";
run;
quit;

ods graphics off;

%END;  /* End of the do loop for i = 1 to the last i */
%MEND;  /* End of the make_print MACRO */

/* Run make print MACRO */
%make_print();
```

This code produces a cascade of JPEGs in an HTML file that shows each student's data against the group's benchmarks. For this group, the lowest student attendance was 158 days and the highest was 179 days. The first quartile was at 171 days and the third quartile was at 177 days. Thus, we can say Donna Marley's 173 days of attendance was not that unusual, but Edgar Ferrel's 164 days was unusually low. She is in the yellow and he is in the red, as shown in the following output:





If you wanted to output those reports to individual PDF files, you could do that by redirecting the ODS output to a PDF file. In the preceding code, you need only to substitute several lines. Replace the HTML formatting:

```
ods html style=listing;
goptions reset=all device=javaimg
xpixels=240 ypixels=200;
```

with code redirecting output to individual PDF files:

```
ods pdf style = new style STARTPAGE = NEVER BOOKMARKLIST=NONE
FILE = "C:\Student&name..pdf";
ODS graphics on;
```

When writing a PDF report for an individual student, you could also include any other data on the student such as grades, credits accumulated, discipline action taken, and other relevant information. These instant student report cards could be updated as often as is helpful to students, parents, teachers, and other educators. They could be run as a stored procedure from SAS® BI Dashboard, accessed via a secure password, or exported to PDF for other means of distribution. Finally, instead of generating these KPIs in student name order, you might generate the KPIs in order of lowest attendance or even generate the reports only for students whose data fall in the problematic range.

The preceding examples show how to create a KPI for data where one row represents one observation, but you could obviously do the same using the means of any group against the population benchmarks. For example, we might compare a baseball or cricket team's batting average against all other teams in the league. We might compare the record sales of a greatest hits collection of The Who rock songs using as benchmarks their best, worst, and most typical selling record from the 1970s. In so many different aspects of our personal and professional lives, benchmarked comparisons are the mechanism by which we contextualize data. Benchmarks make data meaningful and facilitate the role of decision makers in acting on those data.

ADDITIONAL USES

Once you are able to generate KPIs, populate the critical values automatically, and run the metric for a large number of unique observations automatically, there are many other ways this basic functionality can be used. The following two examples are worth considering.

First, we have provided examples in which the actual is the observed value and the bounds are set by capturing the quartiles of that distribution. As the bounds are easily assigned, they could be a combination of measures of the observed data distribution and mandated thresholds. For example, we could have set the upper bound for attendance to the legal requirement, 180 days, and used the observed median and Q3 values as other bounds. This might be of great importance where there is an interest in both relative performance and a mandated minimum or maximum level.

Second, although we did not address it here, advanced analytical output can be used to set bounds. Instead of mere means or other basic measures of central tendency, benchmarks can be derived from more sophisticated metrics. For example, bounds could be set using the distribution of factor scores from a factor analysis. If we knew that we could use 36 questions to determine the satisfaction of patients at a hospital and those 36 could be summarized into 6 major factors, we could score the factors using PROC FACTOR and use the resulting factor scores to benchmark high and low levels of satisfaction within different branches of the hospital or even within different clinics. Thus, patient satisfaction could be updated on a daily or weekly basis using the distribution of the scores within a factor. Six dials could be updated weekly to show each clinic how it compares to its peers.

All of that output could be exported from GKPI and displayed on standard Web portals as HTML. That would leave, of course, the challenge of data integration, cleansing, and data security, but the basic work of generating those benchmarked data is made simple with GKPI and the code presented here. To handle the other data work, all of the code presented here can be easily translated into a stored procedure run from the SAS® BI Server. The server with SAS® Data Integration and security features can handle the back-end work with the code presented here generating JPEGs or HTML presented via a portlet. Instead of generating all of the KPIs en masse, a single KPI could be generated on request via an open field filter search or set of menus created using SAS® Enterprise Guide® 4.2 to be run from the SAS BI Server.

CONCLUSION

Many of our observations and experiences are meaningful only in the context of other data, relative to a benchmark. This has always been true. From Shakespeare to the Dow Jones Industrial Average, we make comparisons. Shakespeare asked, “Shall I compare thee to a summer’s day? Thou art more lovely and more temperate.” Every business day at the close of the bell on the New York Times stock exchange, we hear a report of the Dow Jones Industrial Average as either up or down compared to yesterday’s average. Those data are considered in the context of historical trend lines, other economic data, and even other groupings of stocks such as the S&P 500. Data are meaningful only in context.

Using SAS/STAT software and PROC GKPI, SAS users can quickly and efficiently generate dials and other graphical summaries of data against benchmarked performance. SAS users have a great deal of control over the aesthetic aspects of the output, but more importantly over the bounds and other data used to contextualize the metric. SAS users can also take this functionality to production automatically generating the output for a SAS BI Dashboard within a SAS BI Enterprise solution, as JPEG or HTML for another Web-based interface, directly into Word or PowerPoint, or even as PDF files ready to e-mail or print. A few lines of code provide a great deal of power.

REFERENCES

Please see http://support.sas.com for more information, coding manuals, and sample code for PROC GKPI, SYMPUT, and SAS MACRO language used in this paper.

ACKNOWLEDGEMENTS

Special thanks to Tom Edds and Armistead Sapp of SAS for their helpful suggestions and feedback on this paper.

CONTACT INFORMATION

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

Name: Elliot Inman
Enterprise: SAS Institute Inc.

Address: 100 SAS Campus Drive
City, State ZIP: Cary, NC 27513
Work Phone: 919-531-1717
Fax: 919-677-4444
E-mail: Elliot.Inman@sas.com

Name: Michael Drutar
Enterprise: SAS Institute Inc.
Address: 100 SAS Campus Drive
City, State ZIP: Cary, NC 27513
Work Phone: 919-531-1595
Fax: 919-677-4444
E-mail: Michael.Drutar@sas.com