Paper 047-2011

Match Made in Heaven: Predictive Analytics and the SAS® BI Dashboard 4.3

Keith Renison, SAS, Portland, Oregon, United States

ABSTRACT

The use of dashboards to provide critical insight into the past is becoming a standard across virtually all organizations. This importance is made evident by the fantastic evolution of the SAS BI Dashboard, specifically with the release of SAS BI Dashboard 4.3. How then, are analytics integrated into the strategic views of the organization? That is, what role does predictive modeling play in the use of dashboards, and in effective performance measurement?

This paper explores the connection and practical application of predictive analytics in conjunction with data visualizations in SAS BI Dashboard 4.3 to not just view the past, but propel your dashboards into the future.

INTRODUCTION

This paper is divided into three primary areas: “Getting Around,” “Processing,” and “Leading Analytics with Analytics.” Each section contains practical examples of how it’s done (code), and what it looks like (visual). The “Getting Around” section focuses mainly on ways to build navigation that make accessing analytic insight easier for end users. Much of the advice there can also be applied outside of analytic information, and therefore might be useful to anyone using SAS to develop Business Intelligence Dashboard navigation. “Processing” discusses what to think about when targeting the workload, and advises where a given process should run. And last, we’ll explore some creative ways to lead users through analytic discovery to compound the value gained from analytics.

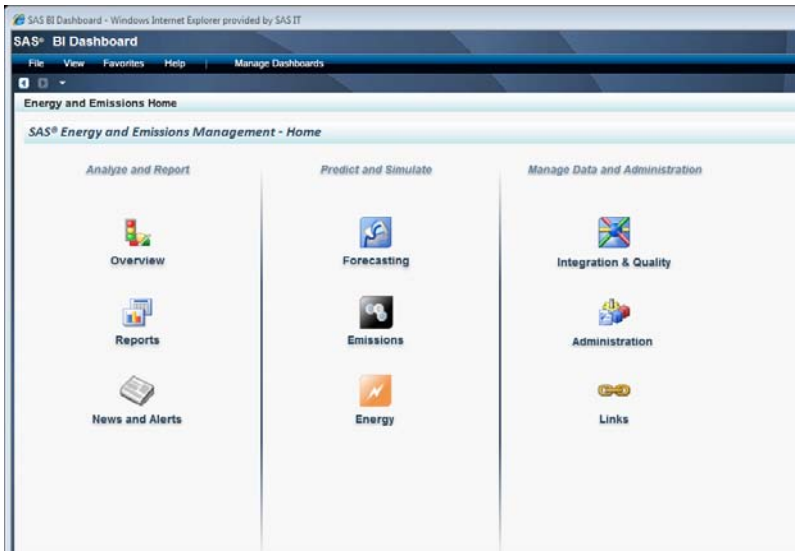
GETTING AROUND

At a dinner party I watched a two-year-old child play with an iPad. She moved around to her different games, picked out one of her favorite movies, and fast-forwarded to her favorite parts of that movie. As I stared with one eyebrow raised, jaw hanging open, I realized there were a couple of characteristics that make these devices so easy for kids, and attractive to adults. She obviously couldn’t read; however she had begun to memorize all the different icons, felt compelled to touch them, and interactively learned what would happen if she did. Though we would like to believe that our minds are more sophisticated, we rarely have the time or patience to learn before we try something. And what’s fun about that? The “What does this button do?” concept is terribly exciting, and when it does what we want, we are compelled to come back for more. You might have the most sophisticated analytic models, but if your end users cannot easily find them or interact with them, they will quickly dismiss their value.

ICONIC STYLE

The world is in love with icons. So, give the world what it wants. Icons are a great way to stain the brain with images that represent tasks. Here are the rules I use for icons: 1) Make them big, but not too big. 2) Label them, but keep labels as short as possible. Remember, users will eventually associate a task to the icon as opposed to what the label says. Therefore, a label is only there to get people started. 3) Pick a standard set of formats and color schemes, shape and creative characteristics such as transparency, reflectivity, shadow, and the like. 4) Configure the size of an icon, or any image for that matter, in an outside application. That is, don’t let the dashboard resize or stretch your image—that will make your images vulnerable to distortion. There is one exception when it comes to borders and lines, which I describe below.

I’ve found one of the best ways to arrange dashboard icons is to start with a main “Home” page, arranging the icons by the type of task. This could either be by type of analysis, or based on some sort of logical workflow. The image below shows an example of such an arrangement:



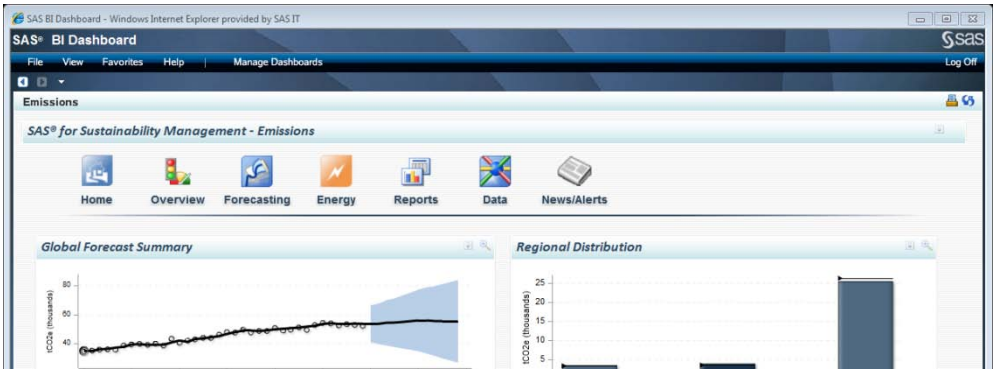
Display 1. Example of Icon-Based Home Screen

Each icon above is either a *.gif or *.png image type, and exactly 48x48 pixels. Each icon is rendered inside a static image content type with a size of 50x50 pixels, with the title removed. These image types are nice because they allow you to set any background or white-space as transparency, which is picked up and rendered by the the dashboard as you would expect in a typical web page. I've found this size to be appropriate for a couple of reasons: 1) going smaller than 50x50 has strange effects on the image, and 2) they're a good size for most people with average sight and are approximately the size of most other computer desktop icons. This might seem obvious, but make sure the icons are evenly spaced, and aligned to each other using the new indicator alignment tools. Misalignment even by a couple of pixels is very easy to spot with the naked eye. Indigestion may occur.

You'll also notice there are vertical lines drawn between sections to cordon off the different areas. These too are a *.png image, set with some fading transparency toward the ends. This one is 12 pixels wide, but can be stretched or shrunk vertically based on your need. This is pretty much the only time I would agree with allowing the dashboard to change the aspect ratio or size of an image. For everything else, define the pixel size in another image editing tool for which it is designed (PowerPoint, Fireworks, etc.). The icon (not the text) is then linked to the relevant dashboard, which brings me to the second concept of navigation.

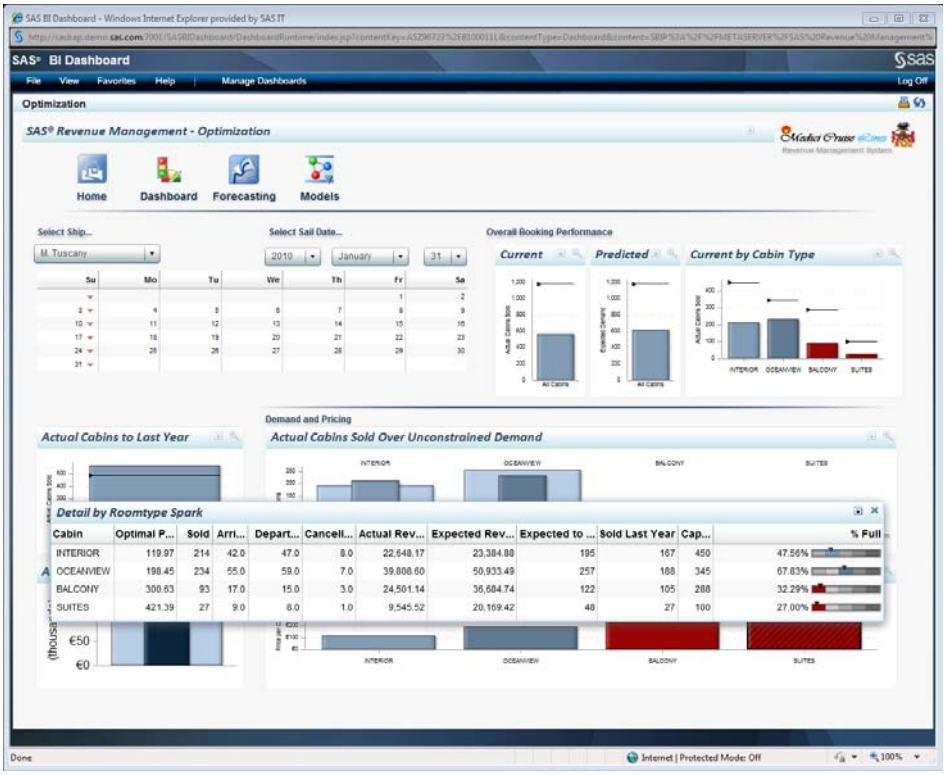
USING YOUR HEAD

The second key to good navigation is to always provide a means to get back to the beginning or to tasks common to the one you're actively in. For this, we create a header bar with the same icons, placing a Home icon always in the same place (typically upper left). If you have too many to fit into a reasonable header, then place only those icons that are meaningful to the current task (users can always go back "Home" if they need to jump to a different area. **Display 2** is an example of this type of header navigation. In this example, I removed the icon for the task I was actively in; however another technique would be to use a "ghosted" or gray representation of that icon when on that dashboard (it could have no link, or link to itself).



Display 2. Example of Header Navigation

It might be argued that using header navigation with large icons uses too much valuable screen real estate. This might have been true with dashboard indicators of static size and detail. However, with the invention of SAS BI Dashboard 4.3, you can pack a lot more detail into a smaller amount of space, and use the "zoom" and mouse-over feature to expand on information that catches your eye, or requires greater exploration. You can also place other indicators behind the first layer, allowing users to click through to other information (linking an indicator to another indicator as shown in **Display 3**). This type of icon-based, drill-through navigation brings a more tactile and intuitive experience to your users.



Display 3. Density, Mouse-over, and Links Enable More Flexibility in the Use of Screen Real Estate

It does take a bit of time to get the first "header navigation" dashboard set up. However, once the dashboard is set up the first time using just the header (no body content), it can be saved as a template and duplicated every time you create your next dashboard. This is done by opening your template and then saving it as a new dashboard with a different name.

Moreover, it might be helpful to number steps when appropriate. That is, if your analytic dashboard requires cascading steps (where one step is dependent upon the selection of something else), you should make it obvious which steps come first, second, and so on. An example of this is given in **Display 7**.

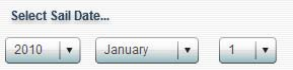
Once the home dashboard, template-header dashboard, and each content-area dashboard is created, you can go back through and link each icon in each dashboard to the appropriate target. And voilà, you've now provided a safe and easy way for even a two-year-old to get around your analytics!

SETTING CONTEXT

> In the measurement world, we set a goal and strive to achieve it. In the universe of possibility, we set the context and let life unfold. ~B. Zander, *The Art of Possibility*

Setting context is important when you combine dashboards and analytics. Consider first the context of time. Dashboards have traditionally been used to gauge the "current state" of the organization. So in many ways, the original time context for a dashboard is always… *now*. The question *"How fast am I going right now?"* is answered by a dashboard in order to make a tactical decision today. Conversely, analytic dashboards tend to be more concerned with the future. So instead we ask *"How fast will I be going tomorrow?"* in order to make a *better*, more strategic decision today. As a result, you will almost always need to provide a way for users to select the context of time for which their decision will be made. Here are a couple of examples:

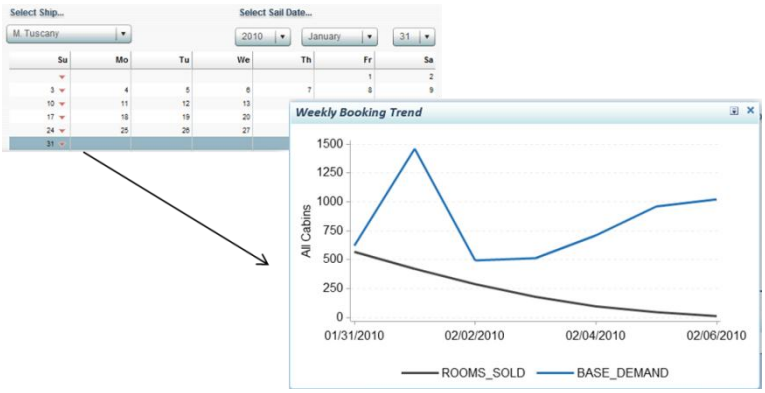
If you're looking to select a specific date, a combination of standard Dynamic Prompts will serve you well (**Display 4**)



Display 4. Selecting a Specific Day

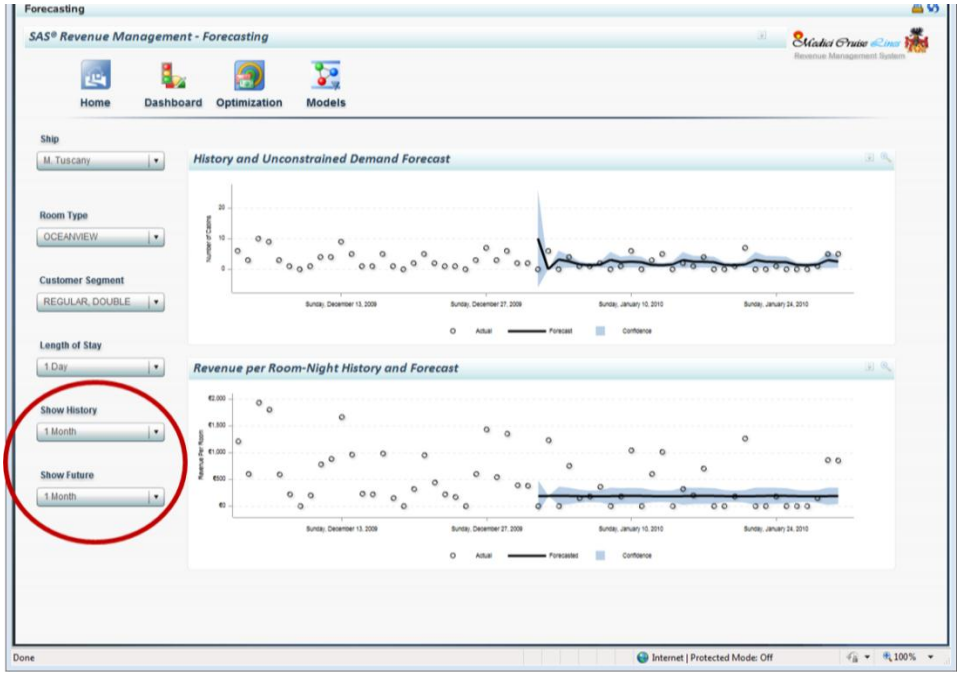
This combination of Year, Month, and Day uses client-side filtering to get to a single day. This "day" Indicator Data model carries with it the actual SAS date value (not displayed) that is passed to the rest of the indicators as a server-side filter, effectively setting the time context for everything else.

To expand on this example, the Year and Month prompts can pass their values into a "calendar" Spark Table Indicator, which filters its data to display a Sunday-Saturday calendar. What is nice about this type of indicator is that if the data is defined correctly, you can render mini performance indicators against a range for each day or week. In a Spark table, the user can also select one of the rows as a link. In this example, the row selection passes the date value context into another linked indicator showing a trend for that week (**Display 5**).



Display 5. Click a Week to Get the Trend for That Week

In other instances, where extensive analytic routines are run in batch that produce sizable data, it might be more appropriate to allow users to subset what they see. In this example, a large multi-dimensional forecast data set is produced in batch (the source table of this one is approximately 500K rows). However, it's probably not necessary for the user to see the entire expanse of history and future values. So providing an ability to select only certain blocks of history or forecast values on either side of "today" (today being the last day of "actual" values) keeps the default query small (30 values on either side), but provides the ability to go back and/or forward up to 12 months.



Display 6. "Show History" and "Show Future" Selections Allow Dynamic Time Subset of Analytic Results

The code that is used for the stored process to subset the data looks like this:

```
/*comment for production*/
/*
%let property_id=1;
%let room_category_id=1;
%let LOS=1;
%let lump_id=1;
%let daysh=30;
%let daysf=30;
%let fcststart=;
*/
libname opt_lib 'D:\RMv1\tenants\Medici Cruise Lines\data\opt_lib';
libname sim_dat 'D:\RMv1\tenants\Medici Cruise Lines\data\sim_dat';
libname tetris 'D:\RMv1\tenants\Medici Cruise Lines\data\tetris';
libname mcl_bi 'D:\RMv1\tenants\Medici Cruise Lines\data\bidatamart';
proc sql noprint;
       select distinct (MIN(date)) into:fcststart
       from mcl_bi.COMB_HIST_FCST
       where forecasted_demand NE .;
quit;
proc sql;
create table work.unqualified_fcst_last as
       select
              date format=date.,
              property_id,
              room_category_id,
              lump_id,
              LOS,
              Actual_rev_per_room format=comma12.,
              actual_rooms_sold format=comma12.,
              price_elasticity format=comma12.,
              forecasted_demand format=comma12.,
              upper_demand format=comma12.,
              lower_demand format=comma12.,
              forecasted_rev_per_room format=comma12.,
              upper_revenue format=comma12.,
              lower_revenue format=comma12.
              from mcl_bi.COMB_HIST_FCST
       where  property_id=&property_id and room_category_id=&room_category_id
                     and lump_id=&lump_id and LOS=&los and date>(&fcststart-
&daysh) and date<(&fcststart+&daysf);
quit;
```

Setting other types of context is also important. These might be organizational, geographic, or dimensional in nature. **Display 6** shows organization (“Ship”) as well as other relevant dimensions. The key here is to work with users to determine how they analyze the results, and give plenty of opportunities to narrow or broaden the context.

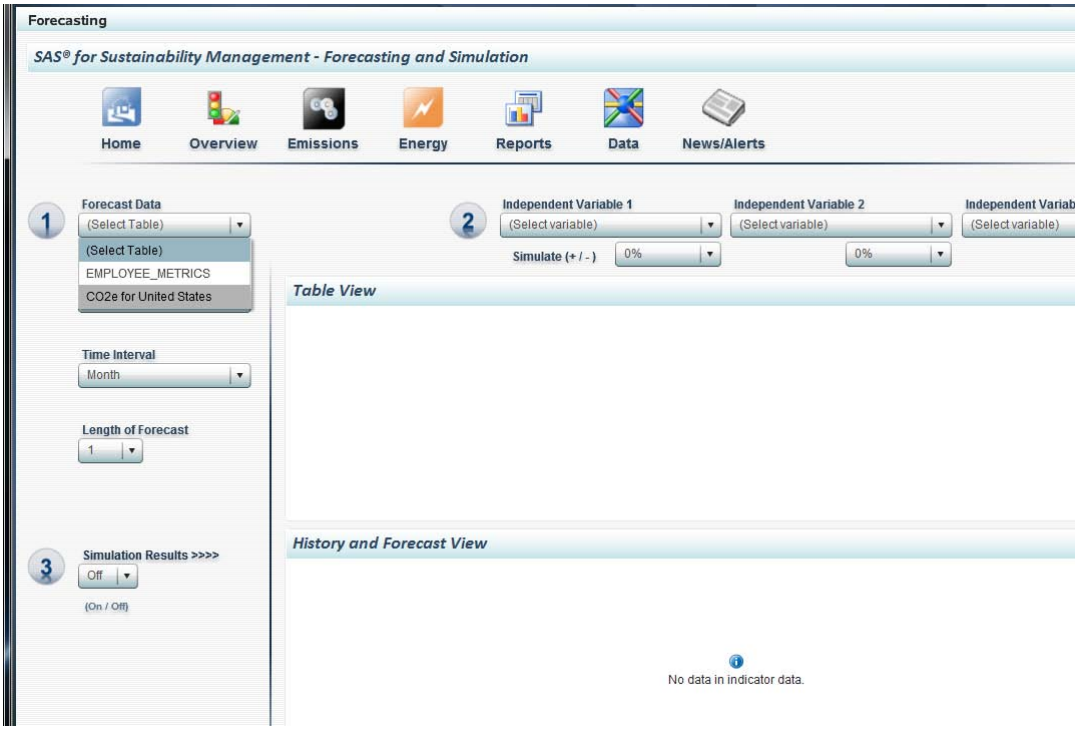
Keep in mind that most SAS BI Dashboard 4.3 graphics can be expanded and further analyzed as a function of the indicator (zoom in, and select the scroll bars on an axis to focus on a given value within the axis).

One of the challenges with dynamic prompts is that they pass their default context whether you like it or not. This is not a big deal with time, since a general focus can be obtained (using time system functions or looking at the analytic data sets). However this becomes more difficult when you require more user context before running a live analytic routine. For example, a what-if forecast that requires a user to select a data set doesn’t necessarily want to have the routine run on all the default values. In this case, it might be necessary to try two techniques. The first is to populate your Dynamic Prompts with default values that return null results. This shortens run time with each change in selection. In this example, the following stored process code produces a default data set with a single row that serves as the dynamic prompt value of the column selection:

```
/*Define the library*/
libname whatif 'D:\whatif';
proc sql;
       create table whatif._default (label='(Select Table)')
       (
```

```
        date num,
        _default num label='(Select variable)',
        _default2     num label='(Select variable)'
        );
quit;
proc sql noprint;
        insert into whatif._default
        values(. 1 1);
quit;
proc contents data=whatif._all_ out=work.whatifmeta noprint;
run;
proc sql;
create table work.datasets as
        select distinct memname,
        (case when memlabel='' then memname else memlabel end) as MEMLABEL
        from work.whatifmeta
        where memname not in ('WHATIF_SIMSTATE',
'WHATIF_INDMULT','INTERVAL','LEAD')
        order by memname desc;
quit;
```

The following Dynamic Prompt with the table labels (or names) is produced:



Display 7. Dynamic Prompts with Default Empty Values

Notice that the default value for the “Forecast Data” prompt is “(Select Table)”. This goes for all the other prompts waiting for user selection in step one.

The other way to control the default behavior is to wrap your main analytic code in a macro that only runs when a user turns it “On.” Notice in step 3 the last thing a user does is to switch the simulation “On” or “Off,” the default selection being “Off.” A data set that looks like this is used to control the Dynamic Prompt:

Name	Value
Off	0
On	1

The forecast simulation stored process code is then wrapped in the following macro:

```
%MACRO whatif;
       %IF &simstate=1 %THEN %DO;
       /*simulation code body is inserted here*/
%MEND whatif;
run;
%whatif;
run;
```

So if the switch is set to "Off," the code simply doesn't run, and the output data sets and subsequent graphics are empty. This use of default null values and on/off switching enables the user to control their preliminary selections before running processes. This also enables a user to change a single variable to have the processes rerun if the switch is left on. Which brings me to the next topic: determining where to run all this stuff.

PROCESSING

As you have seen so far, enabling analytic exploration often requires cascading several prompts and filters using the "Setup Interactions" feature. SAS BI Dashboard 4.3 offers two different types of filters: client-side, and server-side. It's also important to determine *when* your analytic routines will run. That is, which are batch, and which are on the fly. The following sections provide some tips and guidelines.

CLIENT-SIDE FILTERS

Client side filters retrieve all the data specified in the Indicator Data model. That data is then filtered inside the client. This type of filter works best when the full set of data being returned to the client is relatively small. So, pulling a couple of hundred date values for time filtering is relatively fast, and works well when this filtering is performed prior to larger server-side filters. This allows them to load into the view first then pass their values into bigger processes downstream. However, don't get overzealous with the client-side filters. You will be unhappy with your performance if you try to do too much in the client. Instead, look to the server whenever heavy lifting is involved.

SERVER-SIDE FILTERS

I've found that using stored processes with macro variables provides terrific performance paired with tremendous flexibility. For detailed instructions on creating stored processes that can be used by SAS BI Dashboard 4.3, please refer to the *SAS BI Dashboard 4.3: User's Guide*. In general, stored process code is registered in the metadata with default values defined in the macro variables (called "parameters" in the metadata registration). One of the primary goals of the stored process is to reduce the size of the data being returned to the client as much as possible based on the user's selections. Here is the stored process code used to produce the image seen in **Display 6**:

```
%global _ARCHIVE_PATH _ARCHIVE_NAME;
/*begin package body*/
/* parameters comment for production, showing default values */
/*
%let property_id=1;
%let room_category_id=1;
%let LOS=1;
%let lump_id=1;
%let daysh=30;
%let daysf=30;
%let fcststart=;
*/
libname opt_lib 'D:\RMv1\tenants\Medici Cruise Lines\data\opt_lib';
libname sim_dat 'D:\RMv1\tenants\Medici Cruise Lines\data\sim_dat';
libname tetris 'D:\RMv1\tenants\Medici Cruise Lines\data\tetris';
libname mcl_bi 'D:\RMv1\tenants\Medici Cruise Lines\data\bidatamart';
proc sql noprint;
       select distinct (MIN(date)) into:fcststart
       from mcl_bi.COMB_HIST_FCST
       where forecasted_demand NE .;
quit;
proc sql;
create table work.unqualified_fcst_last as
       select
```

```
                date format=date.,
                property_id,
                room_category_id,
                lump_id,
                LOS,
                Actual_rev_per_room format=comma12.,
                actual_rooms_sold format=comma12.,
                price_elasticity format=comma12.,
                forecasted_demand format=comma12.,
                upper_demand format=comma12.,
                lower_demand format=comma12.,
                forecasted_rev_per_room format=comma12.,
                upper_revenue format=comma12.,
                lower_revenue format=comma12.
        from    mcl_bi.COMB_HIST_FCST
        where   property_id=&property_id and
                room_category_id=&room_category_id and
                lump_id=&lump_id and
                LOS=&los and
                date>(&fcststart-&daysh) and
                date<(&fcststart+&daysf);
quit;
/*end package body*/
%macro checkrc(text);
        if rc ne 0 then do;
                msg=sysmsg();
                put msg=;
        end;
        else put "&text succeeded";
%mend;
data _null_;
        rc = 0;
        pid = 0;
        desc = 'unqualified_fcst_last';
        nameV = '';+
        Call package begin(pid, desc, nameV, rc);
        %checkrc(Package init);
        Call insert_dataset(pid, "WORK", "unqualified_fcst_last",
        "unqualified fcst last",'', rc);
        %checkrc(Load Data);
        length fullpath $4096;
        Call package_publish(pid, "TO_ARCHIVE", rc,
        "archive_path, archive_name, archive_fullpath",
        "& ARCHIVE PATH", "&_ARCHIVE_NAME", fullpath);
        %checkrc(Publish);
        call symput('_ARCHIVE_FULLPATH', fullpath);
        Call package_end(pid, rc);
        %checkrc(Package term);
run;
```

I recommend setting up a template for using stored processes for server-side filters. The template is basically two parts. The first is the pre- and post-code processing necessary to place your results in a SAS package. (See above, for example). The second is the actual stored process registered in the metadata. Both components can be produced in advance and copies can be made from them. Each copy, of course, will be slightly changed for each new stored process you need. Second, make sure your default parameter values return some, but very few results. When working with indicator data and indicators in the dashboard builder, the queries are run repeatedly as you step through the building process. If it takes 30 seconds to return your default values, it will take you 15 times as long than if your default values return in two seconds. And lastly, it's sometimes helpful to run your stored process from the SAS Stored Process Web application to make sure it's returning the values you're looking for before building the Indicator Data models.

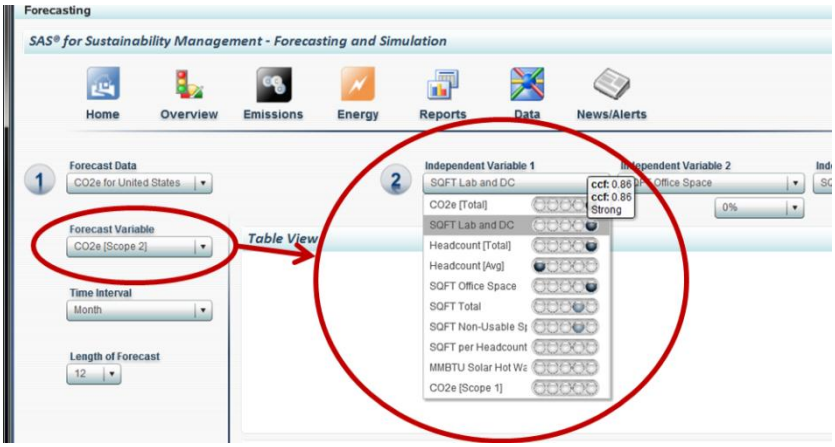
WHEN ANALYTICS ARE RUN

Considerations must be made as to when your analytic routines are executed. For example, the last code was used to query data that was modeled the night before and placed in a reporting data mart. This is because the actual analytic routine involved both optimization and forecasting, which took a large amount of time. These results are also used throughout other reporting interfaces. So in this circumstance it makes most sense to run these major analytic processes in batch, with the creation of a dashboard data mart at the tail-end of those runs. Conversely, when the analytic process is 1) relatively quick based on select parameters, 2) is used for ad hoc simulation, or 3) when the simulation variables are too unknown to pre-calculate, then embedding your analytics within a stored process makes better sense. The following section describes such an example.

LEADING ANALYTICS WITH ANALYTICS

In some cases, the value from analytic analysis can be enhanced with other complimentary analytics. Take for example a forecast, wherein the users are able to select the forecast variable, and then pick from a set of other independent variables. This has often been referred to as "what-if" forecasting, allowing a user to make adjustments to the independent variables to see how they might affect the forecast of the dependent variable. The following example shows how pre-analytics in conjunction with cascading prompts can enable better analysis.

In this example, a user selects a data set and then chooses their forecast variable and lead time. The selection of a dependent variable executes a correlation matrix, populating each of the Independent variable dynamic prompts with the correlation coefficient, sorted in absolute descending order. A range and stop-light indicator is applied to tell the user if the relationship is strong, average, or non-existent. The reason here is to tell the user–in advance of making selections–which variable is most likely to have an impact on their forecast.



Display 8. Selection of Forecast Executes Correlation Matrix of Other Variables in the Data Set

The code that populates the correlation indicator looks like this (with the stored process package code removed):

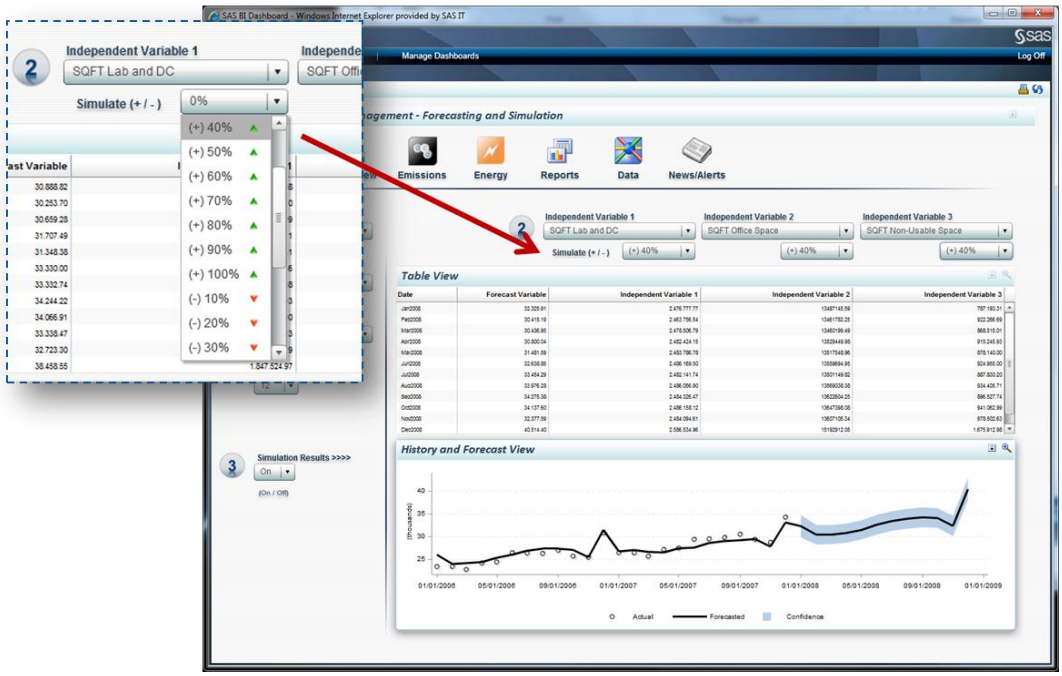
```
/*macros for stored process; commented for production*/
/*%let ds=_default;
%let depvar=_default;*/
libname whatif 'D:\ssm_saas_v1\tenants\demo_tenant\eem\data\analytic';
proc corr data=whatif.&ds(drop=DATE) best=7 outp=data2 NOPRINT ;
var _all_;
with &depvar;
run;
quit;
proc transpose data=work.data2 out=work.data3 name=name;
ID _NAME_;
var _numeric_ ;
run;
proc sql;
create table work.variables_ccf as
      select        (name) as name label='Variable Name',
                    (_label_) as label label='Variable Description',
                    (&depvar) as ccf label='Correlation Coefficient'
      from work.data3
      where UPCASE(name) NE UPCASE("&depvar")
```

```
        order by (abs(&depvar))desc ;
    quit;
```

The users can now select the independent variables while considering how much a given variable will likely influence the outcome (statistically speaking) prior to running the initial forecast. Once all variables are selected, the user can turn on the simulation switch to execute the full simulation. These results are rendered in both a spark table (showing only the future values) and a forecast chart (showing history and forecast values):

Display 9. Full Simulation with Correlation Analysis Leading the Forecast

Following the sample provided on support.sas.com for "Extending the Independent Variable for Multivariate Forecasts," this simulation is using PROC HPF on each of the dependent and independent variables, followed by PROC AUTOREG to factor in the influence of the independent variables. In addition, this simulation goes on to allow the user to override any of the independent variables by a specific percentage (+/-) (**display 10**).



Display 10. Overriding Independent Variables Using a Multiplier with New Forecast Results

As long as the simulation results switch is turned on, the simulation is then automatically re-executed because a variable has changed. Only this time the PROC AUTOREG uses the override values. Below is an example of the code used to produce this simulation:

```
/*begin package code*/
/*macros for stored process; commented for production*/
/*
%let ds=co2e_us_metrics;
%let depvar=total_co2eq_sum;
%let indvar1=lab_sqft_sum;
%let indvar2=headcount_sum;
%let indvar3=hdd;
%let lead=12;
%let interval=month;
%let simstate=1;
%let ind1mult=1;
%let ind2mult=1;
%let ind3mult=1;
*/
%MACRO whatif;
	%IF &simstate=1 %THEN %DO;
libname whatif 'D:\ssm_saas_v1\tenants\demo_tenant\eem\data\analytic';
proc sql noprint;
	select distinct (MAX(date)) into:fcststart
	from whatif.&ds;
quit;
proc hpf data=whatif.&ds out=foreout outfor=test1 lead=&lead;
	id date interval=&interval acc=t;
	forecast &depvar    / select=rmse model=bests accumulate=none;
	forecast &indvar1   / select=rmse model=bests accumulate=none;
	forecast &indvar2   / select=rmse model=bests accumulate=none;
	forecast &indvar3   / select=rmse model=bests accumulate=none;
run;
proc sql;
create table for_depvar as
	select		date,
			(predict) as &depvar label=''
	from test1
	where (upcase(_name_))=(upcase("&depvar"));
quit;
proc sql;
create table for_indvar1 as
	select		date,
			(predict) as &indvar1 label=''
	from test1
	where (upcase(_name_))=(upcase("&indvar1"));
quit;
proc sql;
create table for_indvar2 as
	select		date,
			(predict) as &indvar2 label=''
	from test1
	where (upcase(_name_))=(upcase("&indvar2"));
quit;
proc sql;
create table for_indvar3 as
	select		date,
			(predict) as &indvar3 label=''
	from test1
	where (upcase(_name_))=(upcase("&indvar3"));
quit;
proc sql;
create table combfor as
```

```
        select        for_depvar.date,
                      for_depvar.&depvar,
                      for_indvar1.&indvar1,
                      for_indvar2.&indvar2,
                      for_indvar3.&indvar3
        from   work.for_depvar,
                      work.for_indvar1,
                      work.for_indvar2,
                      work.for_indvar3
        where  for_depvar.date=for_indvar1.date
                      and for_depvar.date=for_indvar2.date
                      and for_depvar.date=for_indvar3.date;
quit;
proc sql;
create table work.foreout1 as
        select        date,
                      &depvar,
                      (case when date > &fcststart then (&indvar1*&ind1mult) else
                      &indvar1 end) as &indvar1,
                      (case when date > &fcststart then (&indvar2*&ind2mult) else
                      &indvar2 end) as &indvar2,
                      (case when date > &fcststart then (&indvar3*&ind3mult) else
                      &indvar3 end) as &indvar3
        from work.combfor;
quit;
proc autoreg data=foreout1 outest=_autoregest;
        model &depvar = &indvar1 &indvar2 &indvar3 / nlag=13 backstep noprint;
        output out=work.foreout2 p=predict ucl=ucl lcl=lcl r=r
        alphacli=0.05 alphaclm=0.05;
run;
proc sql;
        create table work.fcstfinal as
        select Date label='',
        (case when date <= &fcststart then &depvar else . end) as actual
format=comma12.2 label='',
        (predict) as predict format=comma12.2 label='',
        (case when date > &fcststart then ucl else . end) as ucl format=comma12.2
label='',
        (case when date > &fcststart then lcl else . end) as lcl format=comma12.2
label=''
from work.foreout2;
quit;
proc sql;
        create table work.fcstfinal_future as
        select Date label='',
        (predict) as predict  format=comma12.2 label='',
        ucl as ucl format=comma12.2 label='',
        lcl as lcl format=comma12.2 label='',
        (&indvar1) as indvar1 format=comma12.2 label='',
        (&indvar2) as indvar2 format=comma12.2 label='',
        (&indvar3) as indvar3 format=comma12.2 label=''
from work.foreout2
where date > &fcststart;
quit;
        %END;
%MEND whatif;
run;
%whatif;
run;

/*End package code*/
```

CONCLUSION

I hope I have demonstrated using practical examples that the combination of analytics and the flexibility of the SAS BI Dashboard 4.3 can be a powerful tool for visualizing analytic results. I would encourage you to expand and improve upon the concepts presented here as well as contact the author with any questions or suggestions.

REFERENCES

SAS Institute Inc. *SAS 9.2 High Performance Forecasting 3.1: User's Guide*. Cary, NC: SAS Institute Inc. See http://support.sas.com/documentation/cdl/en/hpfug/62015/HTML/default/hpfug_hpfsyntax_sect042.htm.

RESOURCES

SAS Institute Inc. *SAS® BI Dashboard 4.3: User's Guide*. Cary, NC: SAS Institute Inc. See http://support.sas.com/documentation/cdl/en/bidbrdug/63081/HTML/default/viewer.htm#titlepage.htm.

CONTACT INFORMATION

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

Keith Renison
SAS Campus Drive
SAS Institute Inc.
E-mail: keith.renison@sas.com