Paper RV-04

Graphing a Progression of Time Series Plots

Nate Derby, Stakana Analytics, Seattle, WA
Laura Vo, Stakana Analytics, Seattle, WA
Perry Watts, Stakana Analytics, Elkins Park, PA

ABSTRACT

Graphing is an essential step for exploratory data analysis and statistical modeling. However, when graphing an ordered progression of time series plots, it can be difficult to effectively display the progression without looking disorganized and chaotic. This paper shows a couple of approaches to this problem using the `GPLOT` procedure from SAS®/GRAPH® software and the `LAYOUT OVERLAY`, `LAYOUT DATAPANEL` and `SERIESPLOT` statements from the Graphic Template Language (GTL) in ODS statistical graphics.

KEYWORDS: SAS, graph, GTL, time series.

All data sets and SAS code used in this paper are downloadable from `http://nderby.org/docs/RUG11-TSPlots.zip`.

INTRODUCTION: PROBLEMS WITH TIME SERIES PROGRESSION PLOTS

An essential part of statistical modeling is the *exploratory data analysis* (EDA) required to first look at the data and decide which models may be an appropriate fit for the data. This was initially argued by Tukey (1977) and has since become standard practice in statistical modeling. The goal is to look at the data in ways that are conducive to effectively see how the data are truly progressing through time, including large and/or irregularly spaced times.

Time series analysis is the branch of statistics that deals with data indexed by time, where the order of the observations matters. As an example of time series analysis, we look at bookings versus the number of days out for a daily flight from Greater Seattle (SEA) to San Diego (SAN). In this example the variable `daysleft` (days left before departure) acts as a time parameter, `ddate` (departure date) as an independent variable and `bookings` as the response variable. A graph of this quantity is a *cumulative booking curve*, and it is often used in revenue management as a basis for additive and multiplicative pickup models used in statistical forecasting, as explained by Talluri and Van Ryzin (2004, pp. 469-471). However, following the principles of EDA, it can be used to suggest a variety of statistical models.

For our theoretical example, suppose we want to look at the cumulative booking curve for one flight: The one departing on Wednesday, November 3, 2010. We can make a basic plot of this with `PROC GPLOT` within an ODS PDF statement:

```
SYMBOL1 INTERPOL=join MODE=include; ❶

AXIS1 LABEL=( height=1.6 'Days before Departure' ) ORDER=( 0 to 60 by 10 ) ...; ❷
AXIS2 LABEL=( angle=90 height=1.6 'Bookings' ) ORDER=( 0 to 190 by 10 ) ...;

PROC GPLOT DATA=airdata2;
  PLOT bookings*daysleft = ddate / HAXIS=axis1 VAXIS=axis2 VREF=180 LVREF=33 CVREF=black; ❸
RUN;
```

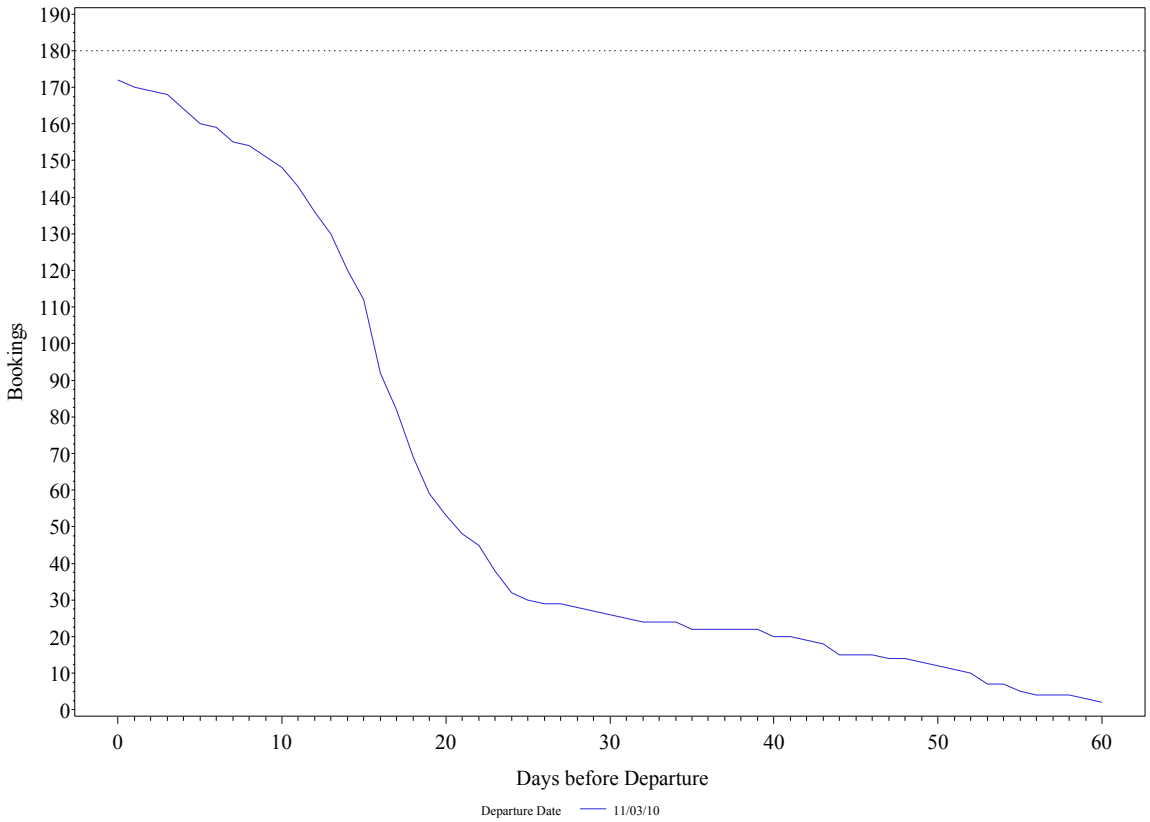
Note the following:

- ❶ The `SYMBOL1` statement tells SAS to connect the points via a line and to include lines drawn toward outliers.
- ❷ The `AXIS` statements explicitly label and order the horizontal and vertical axes.
- ❸ This is our plot, using the `AXIS` and `SYMBOL` statements above. We add a dotted horizontal line at the capacity of 180 seats.

The result is shown in Figure 1(a). This graph is confusing because the horizontal axis represents time running backwards: As we increase the days before departure, we move farther away from the departure date. We will correct for this and align the graph to the right later on in this paper.

A booking curve for one specific flight is not very useful; rather, it is most effective when a series of time series plots are plotted on the same axes, which can be called a *time series progression plot* because it shows the progression of the time series plots.

(a)



(b)

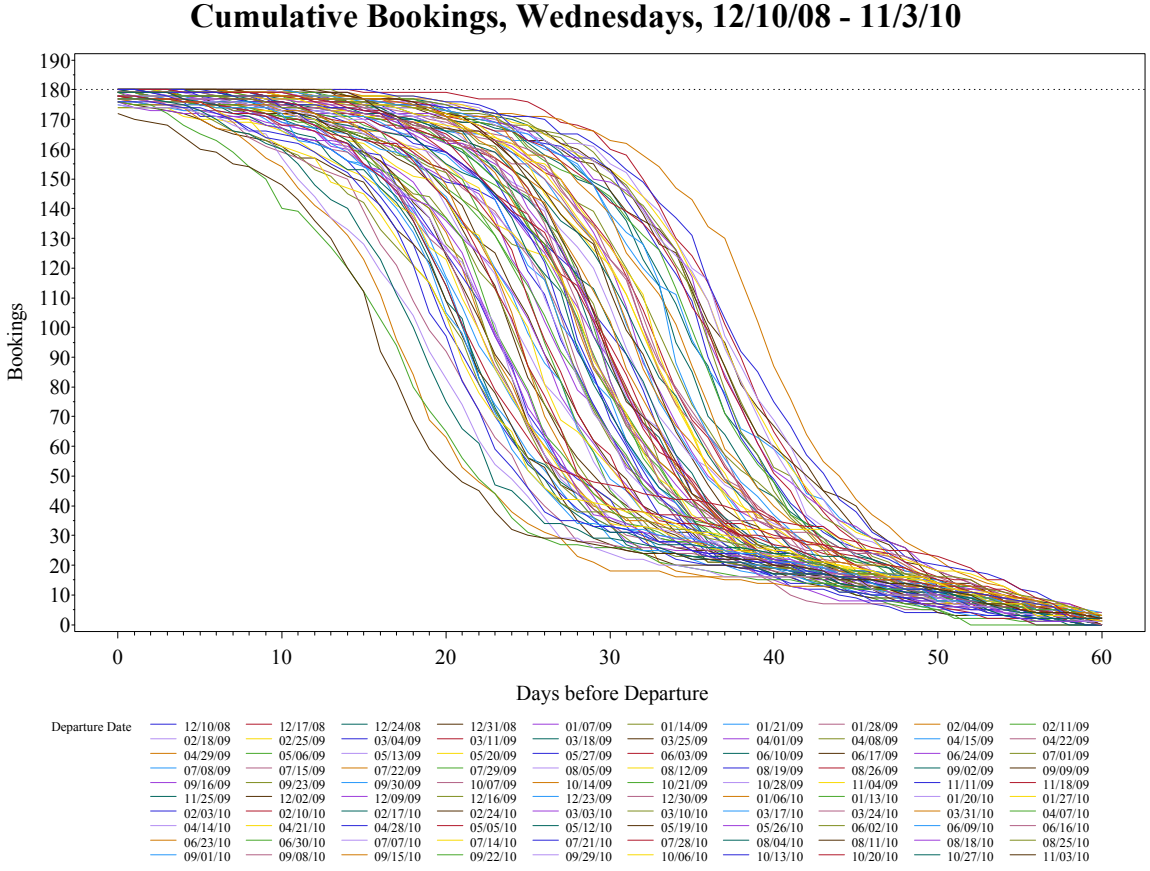


Figure 1: Cumulative bookings curve for the 11/3/10 flight (a) and all Wednesday flights over the past 100 weeks (b), using the default SAS color scheme.

We can produce booking curves for all Wednesday flights (100 of them) by applying the above code to the multi-date data set `airdata` rather than to the single-data data set `airdata2` from before, except that we replace the `SYMBOL` statements with the following:.

```
%MACRO graphData;

  ...
  %DO j=1 %TO &nseries;
    SYMBOL&j INTERPOL=join MODE=include;
    %END;
  ...

%MEND graphData;
```

This uses the macro language[1] to set the remainder of the `SYMBOL` statements to have the same options as `SYMBOL1`, using the default colors that SAS assigns to each time series. `&nseries` is the number of time series that we have (100). Note that the `SYMBOL` statements are indexed by the order of the departure date (`ddate`) in the `PLOT` statement in step ❸. The `SYMBOL`*i* statement iterates to the next value *i*+1 every time the value of `ddate` changes; the actual value of `ddate` is irrelevant. Therefore, the data must be pre-sorted accordingly.

This code produces Figure 1(b). The major problem with this plot, however, is that it fails the central goal of exploratory data analysis: To effectively see what the data are telling us. Indeed, since there is no natural order of the colorings, it is very difficult to visualize the order of the progression. SAS intentionally sets these default colors to maximize contrasts between each progression, since it is designed to make it easy to differentiate the different progressions. However, we are now attempting to minimize the contrast between individual time series plots so that we can visualize general tendencies of the overall progression. Furthermore, with so many time series plots at once, with repeating or very similar colors, the legend is a distraction which serves no useful purpose.

With this distorted view of the data, a statistician might choose to fit a random intercept model, similar to that illustrated in Diggle et al. (2002, p. 136),[2] since it looks like the intercepts are random. But *this kind of model would be inappropriate*, since the intercepts actually follow an ordered progression, as we'll see in the next section.

A BETTER TIME SERIES PROGRESSION PLOT

For exploratory data analysis, the key to making a better time series progression plot is to put a visual order to the colorings. We can do this by picking a color and making the shade lighter for older time series and darker for more recent ones. Then the plot would look similar to a contour plot, except that their contours could cross. For this example, blue is used, since that color can easily be seen from both black-and-white and color printers. However, the ideas in this paper can be applied to any color.

We can use the exact same code we used before, but with different shades of blue chosen for the `COLOR` option in the `SYMBOL` statements. For coding up those shades of blue, we need to learn a little about how SAS codes colors. Watts (2001, 2002, 2003, 2004) has written a series of detailed papers about working with colors within SAS ODS. As best explained in her 2004 paper and briefly summarized here, a color is uniquely identified by the percentage of the primary additive colors red, green and blue. Each color is given a number between 0 and 255, which is the full range of an 8-bit byte (since 2^8 = 256 different values). Many computing applications and formats such as JPG and TIFF code colors as a triplet of these values. Thus,[3]

Blue is (0,0,255), White is (255,255,255), Black is (0,0,0).

SAS codes these colors as hexadecimal values, so that each character represents an integer from 1 to 16 (integers 10 to 16 are represented by the letters A through F, respectively). Each hexadecimal digit is multiplied by a power of 16, with the rightmost digit being multiplied by 16^0, the second rightmost digit being multiplied by 16^1, and so on. For example, a value of 213 would be coded as D5, since D represents the integer 13 and

$$(13 \times 16^1) + (5 \times 16^0) = 208 + 5 = 213.$$

Fortunately, SAS has a hexadecimal format for us, `hex2.` To convert decimal 213 into hexadecimal, for example, just enter `PUT( 213, hex2. )`, which gives us `D5`. A color code in the RGB system combines three hexadecimal numbers into CX*RRGGBB*, where *RR*, *GG* and *BB* represent the two-digit values for red, green and blue components. Thus,

Black is CX000000, Royal Blue is CX0000FF, Light Blue is CXB2B2FF, White is CXFFFFFF.

[1] This macro language won't work in open code; it must be within a macro.
[2] The model illustrated on that page is a logistic regression model, which would be inappropriate here. However, the random intercept component is comparable.
[3] For a color map, see `http://en.wikipedia.org/wiki/Rgb#The_24-bit_RGB_representation`.

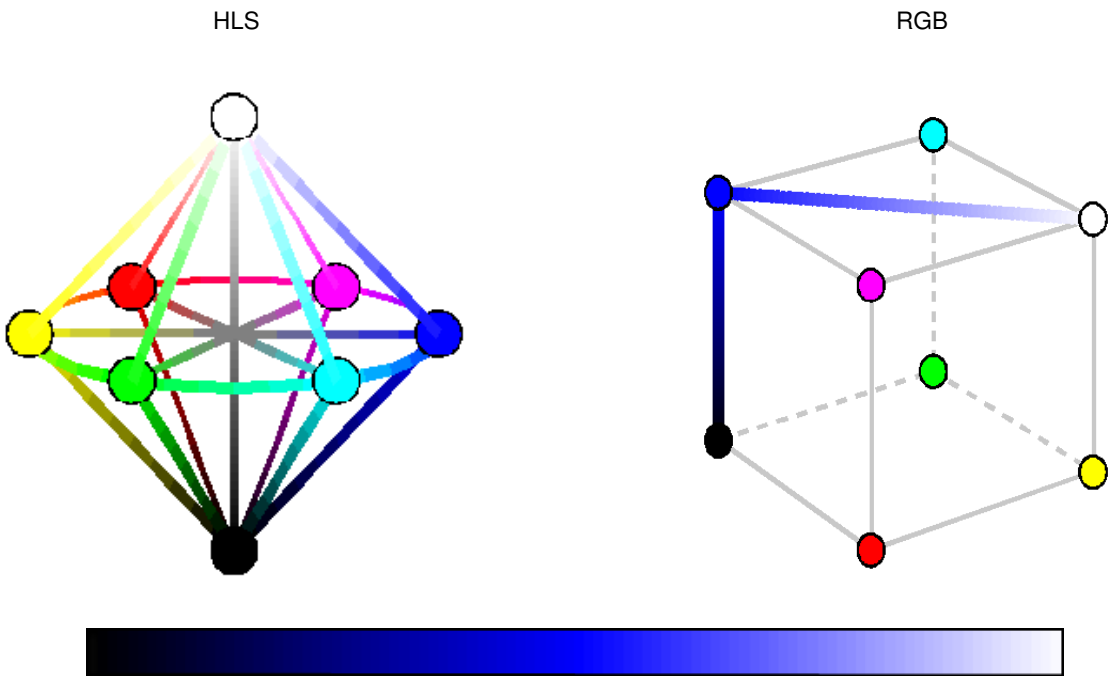


Figure 2: A pictorial definition for the HLS and RGB color spaces shows how the blue lightness scale in HLS maps from black to white in RGB. Fully saturated colors appear at the surface of all color spaces. Interior grayish shades are unsaturated.

However, the HLS coding systems is better suited for working with lighter and darker shades of the same color. The format for HLS is H*HHH****LL****SS* where *HHH* is a three-digit hexadecimal value for hue (0-360 Decimal, 0-168 Hex), ***LL*** is a two-digit hexadecimal value for lightness (0-FF), and *SS* is again a two-digit hexadecimal value for saturation (0-FF). Thus,

Black is H000**00**FF, Royal Blue is H000**80**FF, Light Blue is H000**D9**FF, White is H000**FF**FF.

To get a lighter or darker shade of the same hue in HLS, only the lightness component has to be changed. A visualization of the difference between the HLS and RGB systems is shown in Figure 2. HLS translates to a double-ended cone whereas the RGB space is a cube.

For our time series progression plot, we can plot the first time series in white (H000**FF**FF), the last one in black (H000**00**FF), and the intermediate ones in the equidistantly calculated shades of blue. Assuming we have n time series progressions, in pseudocode we have

FOR $j = 1,\ 2,\ \ldots,\ n$,

LET $k = \left\lfloor 100 \cdot \left(\frac{n-j}{n-1} \right) \right\rfloor$ ❶

SYMBOLj COLOR = HLS(hue=0%, lightness=k%, saturation=100%) in RGB ❷

END

A couple notes:

- ❶ The mathematical symbol $\lfloor \cdot \rfloor$ means the *floor* of a number, which is the greatest integer less than or equal to that number. For instance, $\lfloor 5.5 \rfloor = 5$ and $\lfloor 6 \rfloor = 6$. We use this to ensure that we have an integer between 0 and 100. The $\frac{n-j}{n-1}$ term tells us what percentage to use; as j increases from 1 to n, this expression goes from 100 (giving us H000**00**FF (white) in step ❷) to 0 (giving us H000**00**FF (black) in step ❷).
- ❷ Here we code the color first into HLS, then into RGB (since ODS only accepts RGB values).

To code an HLS value in SAS, we can use the macro `%HLS`, part of the `%COLORMAC` set of macro utilities introduced in version 9 SAS. These macros simplify the translation of color codes from decimal to hexadecimal and from one coding system to another. Anticipating that ODS won't accept HLS values, we convert them now to their RGB counterparts using the SAS macro `%HLS2RGB` (also part of `%COLORMAC`). Below is the SAS code that implements the pseudocode above:

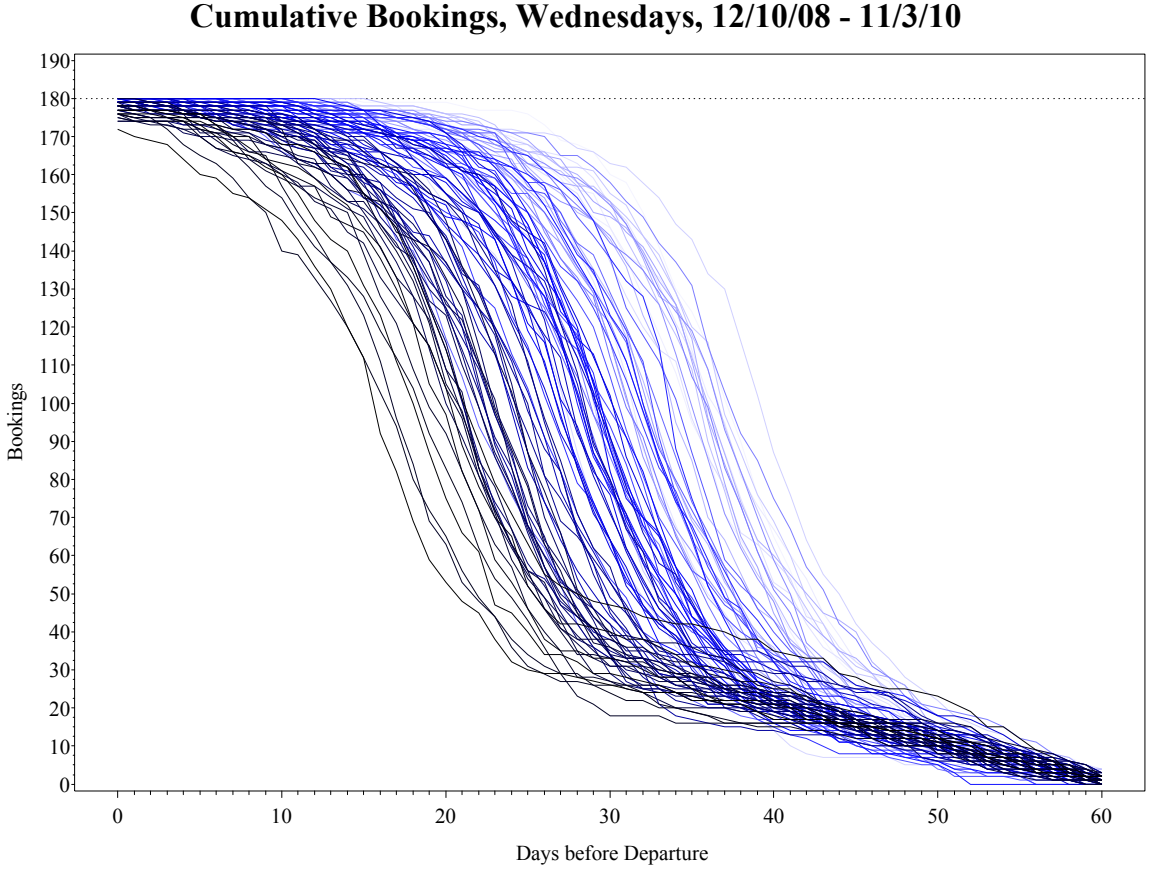


Figure 3: Cumulative bookings curves for all Wednesday flights over the past 100 weeks, using the improved color scheme.

```
%MACRO graphData;

  ...
  %COLORMAC;   ❶
  %DO j=1 %TO &nseries;
    %LET k = %SYSFUNC( FLOOR( 100*( &nseries - &j )/( &nseries - 1 ) ) );   ❷
    SYMBOL&j INTERPOL=join MODE=include COLOR=%HLS2RGB( %HLS( 0, &k, 100 ) );   ❸
    %END;
  ...

%MEND graphData;
```

❶ `%COLORMAC` is an umbrella macro that contains the submacros `%HLC` and `%HLS2RGB`.

❷ The macro `%SYSFUNC` is required to run the function `FLOOR` within a macro step.

❸ The color conversion macros are macro functions. Like any function, they can be nested. The inner-most macro, HLS, converts an HLS code from decimal percent into hexadecimal. HLS2RGB then does what the name implies: it takes an HLS hex code and translates it into RGB.

Again, the `SYMBOL`*i* statements iterate whenever a new value of `ddate` is reached in the `PLOT` statement (step ❸ on page 1). For this to work, the data must be sorted by departure date (`ddate`).

The result is shown in Figure 3. Here we see something we couldn't see with our multicolored plot in Figure 1(b): As the departure date increases, customers wait longer before buying tickets (i.e., it shifts to the left, so that customers generally make their bookings on a smaller number of days before departure). Specifically, our random intercept model that we considered using in our previous exploratory data analysis (Figure 1(b)) would be inappropriate, since we have a *deterministic* intercept (depending on departure date) rather than a *random* intercept.

While Figure 3 is good for finding an appropriate statistical mode, it still falls short on usability for deeper visual analysis. We're showing two years worth of data here, but we don't have a clear sense of what booking curves correspond to what dates. Naturally we can't label every series as in Figure 1(b), but our solution in Figure 3 doesn't show *any* dates. A better solution is to put the data into different groups ordered by time of departure and to order the colorings of them.

For the color, as shown in Figure 3, we don't want to start with complete white that is impossible to see or end with black that has no detectable hue. So for the lightness, instead of going from 100% (white) to 0% (black), we can go from 85% to 15% in increments of 15% (with the first increment being 10% for the math to fit):

```
%MACRO AssignBlues;

  %COLORMAC;
  %LET lite=85;
  %DO i = 1 %TO 6;
    %LET color&i = %HLS2RGB( %HLS( 0, &lite, 100 ) );
    %IF &i eq 1 %THEN %LET lite = %EVAL( &lite - 10 );
      %ELSE %LET lite = %EVAL( &lite - 15 );
  %END;

%MEND AssignBlues;
```

Once again, we need `%COLORMAC` to make use of the macro functions `%HLS2RGB` and `%HLS`. This gives us six shades of blue.

For our graph, we will instead use the Graphics Template Language (GTL) to get a slightly different view. This is discussed extensively in Kuhfeld (2010). We will list the code below with minimal details, with most of the style attributes removed. Macro variable assignments have also been resolved.

```
PROC TEMPLATE;
  DEFINE STATGRAPH seriesplt;
    DYNAMIC title title2;
    BEGINGRAPH / DESIGNWIDTH=600 DESIGNHEIGHT=350 ...;
      ENTRYTITLE title;
      ENTRYTITLE title2;
      LAYOUT OVERLAY / YAXISOPTS=( GRIDDISPLAY=on ❶
                         LABEL="Bookings"
                         LINEAROPTS=( TICKVALUELIST=(0 30 60 90 120 150 180)
                         VIEWMIN=0 VIEWMAX=180 ) )
                       XAXISOPTS=( LABEL="Days Before Departure"
                         LINEAROPTS=( TICKVALUESEQUENCE=( START=0 END=60 INCREMENT=10 ) ) );
        SERIESPLOT Y=y1 X=x1 / GROUP=ddate ❷
                       LINEATTRS=( COLOR=CX00004B PATTERN=1 THICKNESS=1 )
                       YAXIS=y; ❶
        ...
        SERIESPLOT Y=y6 X=x6 / GROUP=ddate
                       LINEATTRS=( COLOR=CXB2B2FF PATTERN=1 THICKNESS=1 )
                       YAXIS=y;
        LAYOUT GRIDDED / COLUMNS=1 AUTOALIGN=( topleft topright ) ❸
                       OPAQUE=true BACKGROUNDCOLOR=CXF5F5F5 BORDER=true;
          ENTRY HALIGN=left "Departure Date Range";
          ENTRY HALIGN=left TEXTATTRS=( WEIGHT=bold COLOR=CX00004B ) "12/10/08-04/05/09";
          ...
          ENTRY HALIGN=left TEXTATTRS=( WEIGHT=bold COLOR=CXB2B2FF ) "07/13/10-11/06/10";
        ENDLAYOUT; /*GRIDDED*/
      ENDLAYOUT; /*OVERLAY*/
    ENDGRAPH; /*END GRAPH BLOCK*/
  END;
RUN;
```

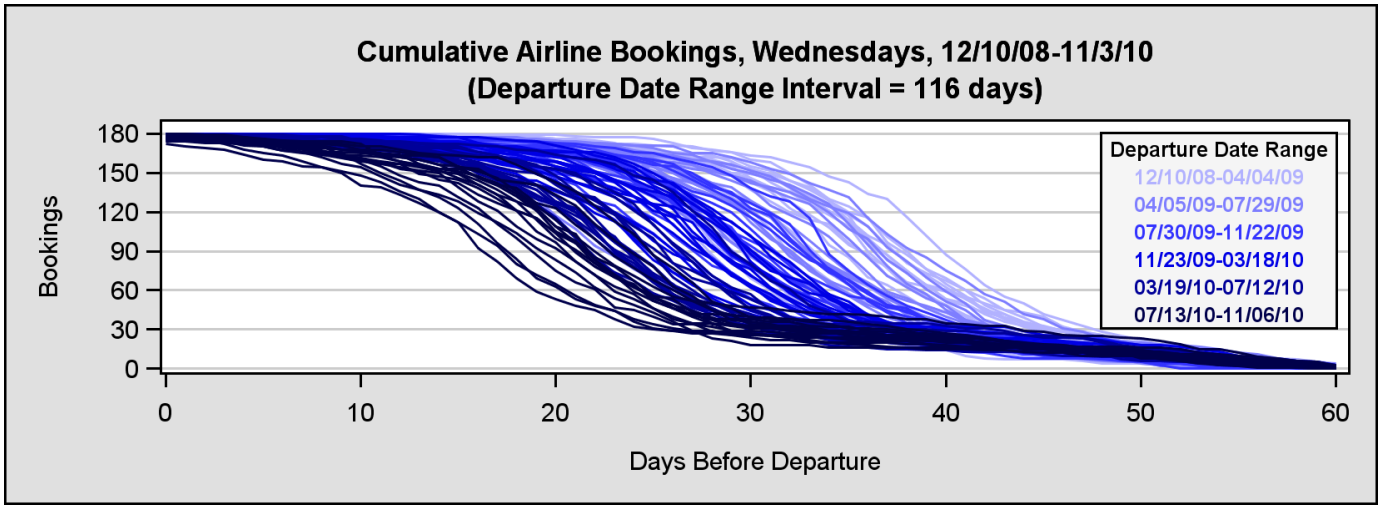


Figure 4: Cumulative booking curves for 100 consecutive Wednesday flights using the improved color scheme.

As outlined above, ODS is hierarchical with nested functionality. At the outermost level, a `STATGRAPH` template named `seriesplt` is defined. Dynamic variables that bear a similarity to macro variables are declared at this point. Next, graph size and titles are entered with `BEGINGRAPH`, followed by `LAYOUT OVERLAY` that associates axes properties with a single plotting area. In this particular example, six series plots are inserted sequentially into the defined plotting area. Each new series plot overlays the one previously created. Thus, the light early plots are partially hidden by the darker plots that follow. Finally `LAYOUT GRIDDED` (contained within `LAYOUT OVERLAY`) holds the graph's legend. Some further details:

❶ Axes options are defined first in the `LAYOUT OVERLAY` statement. It is possible to replace the Y axis with the Y2 axis in this graph. Therefore either `YAXISOPTS` or `Y2AXISOPTS` are specified here and with links defined in subsequent `SERIESPLOT` statements with `YAXIS=y` or `YAXIS=y2`. When `y2` is specified, then the X axis is reversed with `REVERSE=true` (shown in a later example).

❷ 100 Wednesday series plots are created, because `GROUP=ddate` (departure date). However, color is assigned by *six* date *ranges* of 116 days each. Color assignment occurs by subdividing data at the date range level. `daysleft` (Days before Departure) is used to for creating variables X1-X6 whereas the corresponding Y1-Y6 variables come from `bookings`.

❸ `LAYOUT GRIDDED` rather than a `DISCRETELEGEND` statement is better suited for color labeling. Legend line colors wash out because line widths fixed at 1 pixel are too narrow. Bold font used for specifying date ranges in the `GRIDDED` layout is much wider. `AUTOALIGN=( TOPLEFT TOPRIGHT )` also comes in very handy. When the Y axis is being used, the legend needs to be `TOPRIGHT`, whereas `TOPLEFT` is automatically assigned for the Y2 axis. ODS graphics is set up to avoid data/legend collisions.

The end result is shown in Figure 4.

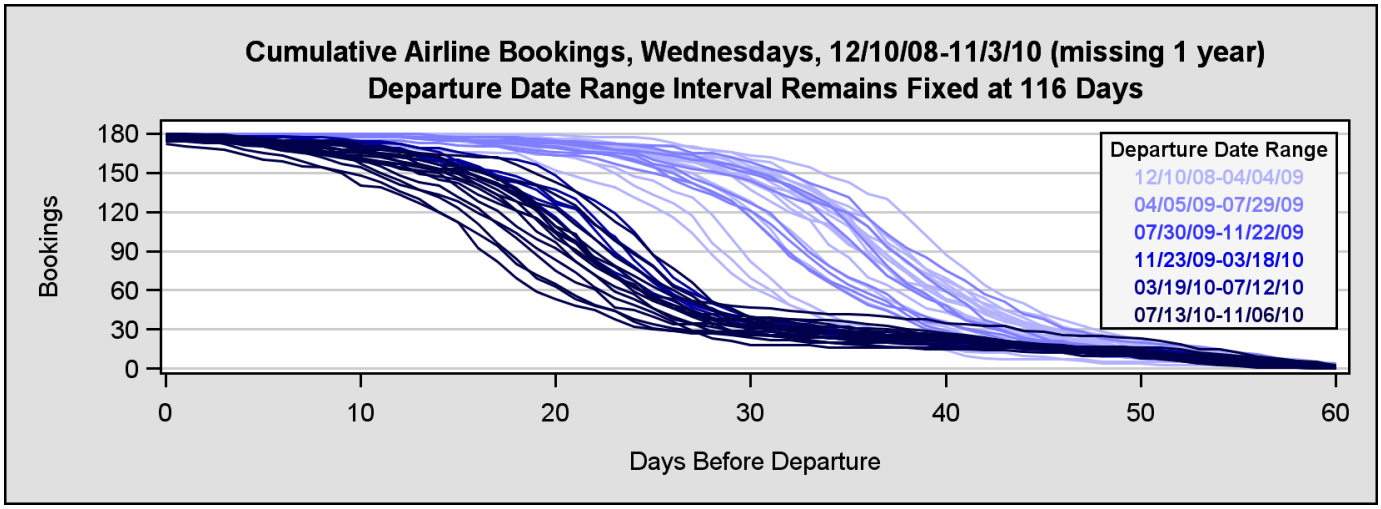


Figure 5: Cumulative booking curves with missing data are displayed. By comparing the legend to the plot with missing values, color discontinuities can easily be identified.

PROGRESSION PLOTS WITH IRREGULARLY SPACED DATES

The `seriesplt` template created above can easily handle irregularly spaced progressions. Only the input data set needs to be changed. Let's suppose, for example, that a year's worth of bookings data turns up missing. Instead of deleting an entire observation that includes departure date and the number of days before departure, just retain these variables and set `bookings` to missing:

```
IF ddate >= MDY( 6, 1, 2009 ) AND ddate <= MDY( 6, 1, 2010 ) THEN bookings = .
```

Missing values from `bookings` will automatically transfer to Y2-Y5. By comparing departure date ranges to the year cut-off dates, it can be determined that the contents of Y1 and Y6 will remain untouched whereas all the values in Y3 and Y4 will be set to missing. Y2 and Y5 fall somewhere in the middle with some but not all values being set to missing. The log reflects this outcome by issuing the following warning messages:

```
WARNING: Y=Y3 is invalid. The option expects at least one non-missing value in the column.
WARNING: Y=Y4 is invalid. The option expects at least one non-missing value in the column.
```

Setting the Y-coordinate to missing takes advantage of the fact that nothing is plotted when a single coordinate is missing from the output data set. Thus Figure 5 displays cumulative booking curves with data removed from the year in the middle while the legend acts as if it were dealing with a complete data set.

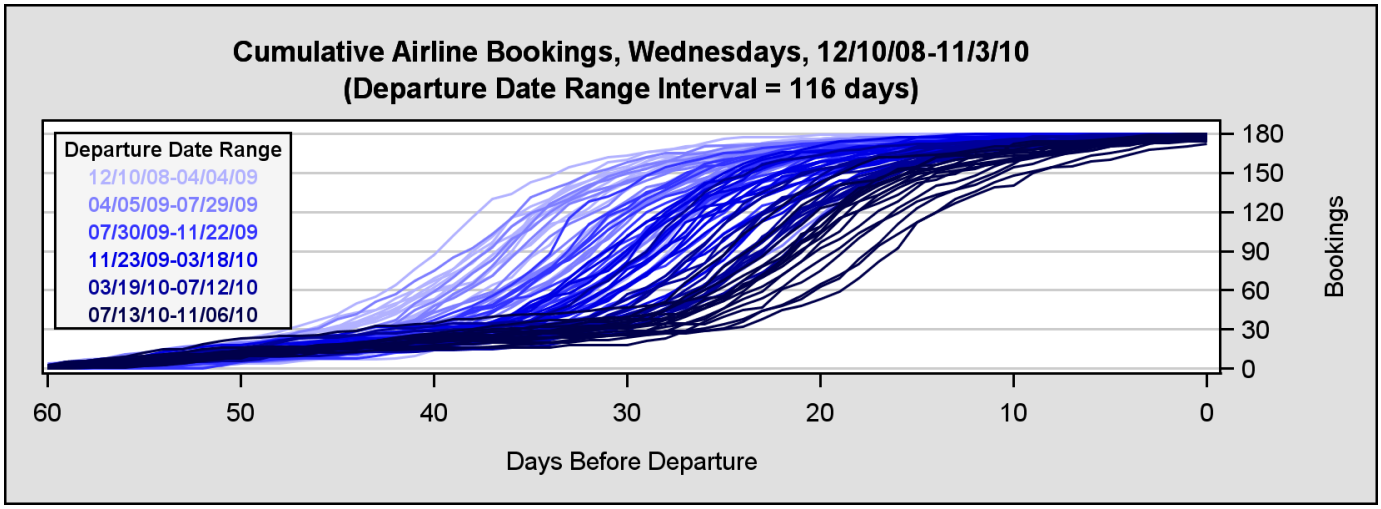


Figure 6: Cumulative bookings for 100 consecutive Wednesday flights aligned to the right.

PROGRESSION PLOTS ALIGNED TO THE RIGHT

As mentioned in the introduction, our horizontal axis is going in the wrong direction. Again, the way it is now, time runs backwards as the horizontal axis increases. It's much more intuitive to have the graph the other way around, such that the days before departure decreases to zero. Since the curves increase to the upper right, it's more natural to label the vertical axis on the right rather than on the left. This is the standard way of formatting the graph in the revenue management industry, as shown e.g., in Talluri and Van Ryzin (2004, p. 471).

Doing this with `PROC GPLOT` is actually quite involved, as shown in Derby and Vo (2010) (an earlier version of this paper). However, it's actually quite easy with GTL, as we need only change a few parameters of the `seriesplt` template as noted below:

```
PROC TEMPLATE;
  DEFINE STATGRAPH seriesplt;
    ...;
    BEGINGRAPH / ...;
      ...;
      LAYOUT OVERLAY / Y2AXISOPTS=( ... )   ❶
                       XAXISOPTS=( ... REVERSE=true );   ❷
        SERIESPLOT Y=y1 X=x1 / GROUP=ddate ... YAXIS=y2;   ❶
        ...
        SERIESPLOTY=y6 X=x6 / GROUP=ddate ... YAXIS=y2;   ❶
        /* LAYOUT GRIDDED is the same */
      ENDLAYOUT;
    ENDGRAPH;
  END;
RUN;
```

❶ The Y2 axis is the right vertical axis. Therefore, we use `Y2` rather than `Y` in the indicated places.

❷ `REVERSE=true` reverses the order of the horizontal axis.

USING A `DATAPANEL` LAYOUT TO DISPLAY CUMULATIVE AIRLINE BOOKINGS

According to SAS Institute (2009, p. 59), the `LAYOUT DATAPANEL` statement in GTL is used to create "a grid of graphs based on one or more classification variables and a graphical prototype". With `LAYOUT DATAPANEL`, all cells have a uniform definition that is specified in a single subordinate `LAYOUT PROTOTYPE` statement. The restriction imposed on the output by `LAYOUT PROTOTYPE` means that panels must be colored the same. Color isn't really a necessity in this instance, however, since each group is assigned to a different panel. Again, the `REVERSE` option is supported in `LAYOUT DATAPANEL` as it was in `LAYOUT OVERLAY`. The code for the `DATAPANEL` graph in Figure 7 is shown on the next page.

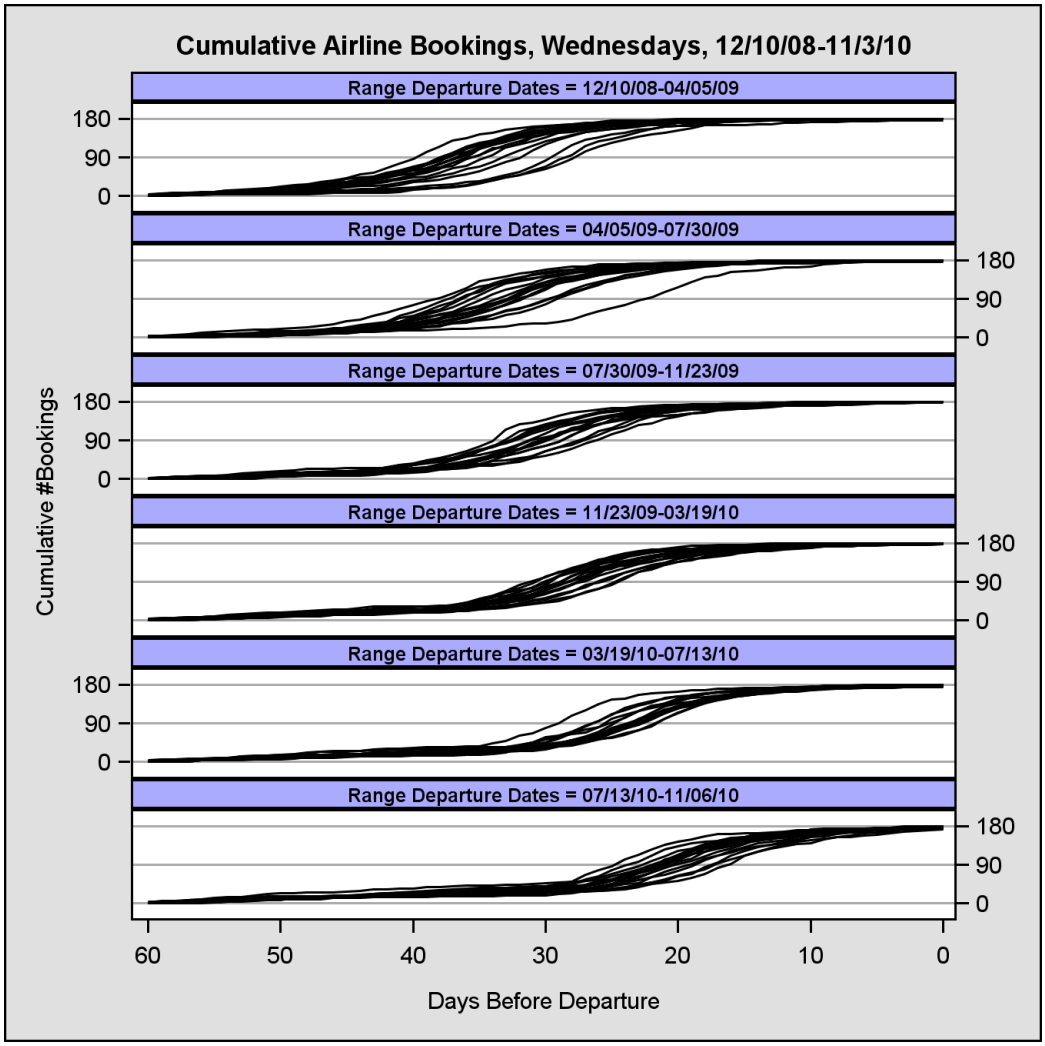


Figure 7: Cumulative bookings for 100 consecutive Wednesday flights are represented in a DATAPANEL graph. Outliers are more easily identified, since data overlay is greatly reduced in a multi-cell display. The relationship between booking lead times and departure dates is also well established in this plot. On average, it appears that flyers in the beginning of 2009 booked their reservations 15 days earlier than they did in the latter part of 2010.

```
PROC TEMPLATE;
  DEFINE STATGRAPH seriesplt;
    DYNAMIC TITLE;
    BEGINGRAPH / DESIGNWIDTH=450 DESIGNHEIGHT=450 BORDERATTRS=( THICKNESS=1px );
      ENTRYTITLE TITLE / TEXTATTRS=( SIZE=8pt );
      LAYOUT DATAPANEL CLASSVARS=( byddatelbl ) / COLUMNDATARANGE=union
          HEADERLABELATTRS=( WEIGHT=bold SIZE=6px ) HEADERBACKGROUNDCOLOR=CXAAAAFF
          ROWAXISOPTS=( GRIDDISPLAY=on DISPLAY=all ALTDISPLAY=none DISPLAYSECONDARY=none
          ALTDISPLAYSECONDARY=( ticks tickvalues )
          LINEAROPTS=( TICKVALUESEQUENCE=( START=0 END=180 INCREMENT=90 ) ) )
          COLUMNAXISOPTS=( LABEL="Days Before Departure" REVERSE=true
            LINEAROPTS=( TICKVALUESEQUENCE=(START=0 END=60 INCREMENT=10 ) ) );
        LAYOUT PROTOTYPE;
          SERIESPLOT Y=bookings X=daysleft / GROUP=ddate LINEATTRS=( PATTERN=1 COLOR=black );
        ENDLAYOUT; /*PROTOTYPE*/
      ENDLAYOUT; /*DATAPANEL */
    ENDGRAPH; /*END GRAPH BLOCK*/
  END;
RUN;
```

CONCLUSIONS

Underlying patterns in a progression of time series plots can be revealed graphically when a palette of related, ordered colors is applied to the data that are being displayed. The palette is easily defined using SAS software color conversion macros. The advantages of ODS statistical graphics have also been illustrated in this paper. Both legend placement and axes reversal can be implemented with ease, and the new `DATAPANEL` plot greatly reduces data overlay in the output. Whether the graphs are produced with traditional SAS/GRAPH software or ODS statistical graphics, special care must be taken to insure that irregularly spaced data are displayed correctly.

REFERENCES

Derby, N. and Vo, L. (2010), Graphing a progression of time series plots, *Proceedings of the 2010 Western Users of SAS Software Conference*.
`http://www.wuss.org/proceedings10/analy/2954_6_ANL-Derby.pdf`

Diggle, P. J., Heagerty, P., Liang, K.-Y. and Zeger, S. L. (2002), *Analysis of Longitudinal Data*, second edn, Oxford University Press, Oxford, UK.

Kuhfeld, W. F. (2010), *Statistical Graphics in SAS: An Introduction to the Graph Template Language and the Statistical Graphics Procedures*, SAS Institute, Inc., Cary, NC.

SAS Institute (2009), *SAS/GRAPH 9.2 Graph Template Language Reference*, SAS Institute, Inc., Cary, NC.

Talluri, K. T. and Van Ryzin, G. J. (2004), *The Theory and Practice of Revenue Management*, Kluwer Academic Publishers, Boston.

Tukey, J. (1977), *Exploratory Data Analysis*, Addison-Wesley, Boston.

Watts, P. (2001), Defining colors with precision in your SAS/GRAPH application, *Proceedings of the Fourteenth Northeast SAS Users Group Conference*.
`http://www.nesug.org/proceedings/nesug01/cc/cc4023.pdf`

Watts, P. (2002), Using ODS and the macro facility to construct color charts and scales for SAS software applications, *Proceedings of the Twenty-Seventh SAS Users Group International Conference*, paper 125-27.
`http://www2.sas.com/proceedings/sugi27/p125-27.pdf`

Watts, P. (2003), Working with RGB and HLS color coding systems in SAS software, *Proceedings of the Twenty-Eighth SAS Users Group International Conference*, paper 136-28.
`http://www2.sas.com/proceedings/sugi28/136-28.pdf`

Watts, P. (2004), Advanced programming techniques for working with color in SAS software, *Proceedings of the Twenty-Ninth SAS Users Group International Conference*, paper 091-29.
`http://www2.sas.com/proceedings/sugi29/091-29.pdf`

ACKNOWLEDGMENTS

We thank our clients for working with us on this topic. We also thank our partners/spouses Charles, Hoa and Sam for their patience and support.

CONTACT INFORMATION

Comments and questions are valued and encouraged. Contact one of the authors:

Nate Derby
Stakana Analytics
815 First Ave., Suite 287
Seattle, WA 98104-1404
`nderby@stakana.com`
`http://stakana.com`
`http://nderby.org`

Laura Vo
Stakana Analytics
815 First Ave., Suite 287
Seattle, WA 98104-1404
`lvo@stakana.com`
`http://stakana.com`

Perry Watts
Stakana Analytics
Elkins Park, PA
`pwatts@stakana.com`
`http://stakana.com`
`http://PerryWatts.org`