Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

Calendar-based graphics for visualising people's daily schedules

Earo Wang
Monash University

Di Cook
Monash Univeristy

Rob J Hyndman
Monash Univeristy

Abstract

It is challenging to visualise people's daily schedules as such data are often of multiple seasons and of long time span. One example is Melbourne pedestrian sensor data which are collected at hourly interval and at a number of locations. Owing to human-related activities, the primary depiction of the data features multiple temopral scales such as time of day, day of week, day of year. We develop the `frame_calendar` function that provides a method for rearranging such kind of data in a calendar-based format and accomplishing visualisation with the grammar of graphics, and describe its construction in detail. We provide some examples to demonstrate its usage by applying to the pedetrian data.

Keywords: calendar-based visualisation, temporal-context data.

1. Introduction

This work was originally motivated by studying foot traffic in the city of Melbourne (City of Melbourne 2017). There have been 43 installed sensors counting pedestrians every hour across downtown Melbourne until recently. The dataset can shed lights into understanding people's daily schedules, or assisting administration and business planning. (FIGURE ?: weekday at multiple sensors) Figure (?) lends itself to a number of issues that make exploratory data visualisation challenging in many temporal-context applications involving human behaviours:

1. Variations primarily result from multiple time scales including time of day, day of week, and day of year (such as public holiday and recurred events).
2. Since the data are often collected at daily frequency or a time scale more frequent than daily, they typically involve a large number of observations spanning over a long time period.
3. Measurements of a single type are made at multiple locations at a given time point,

which creates a need in comparing and contrasting between locations.

Calendar-based graphics turn out to be a useful tool in unfolding human-related activities in temporal context (cite). For example, Van Wijk and Van Selow (1999) developed a calendar view of heatmap to represent the number of employees in the work place over a year, where colours indicate different clusters derived from the days. It remarkably contrasts weekdays and weekends, highlights public holiday, and presents other known seasons like school vacations, all of which have influence over the turn-outs in the office. The calendar-based heatmap was implemented in a couple of R packages: **ggTimeSeries** (Kothari and Ather 2016) and **ggcal** (Jacobs 2017). However, these techniques are too constrained to colour-encoding graphics and day of week considered as the smallest time window. Time of day, which serves as one of the most important aspect in explaining variations arising from pedestrian sensor data, will be neglected through daily aggregation. Additionally if simply using colours encoded as quantities, it may becomes less perceivable of the actual variations or behaviours from the viewer's perspective.

We propose a new algorithm via linear algebra tools to go beyond the calendar-based heatmap. The approach is developed with these conditions in mind: (1) to make time of day present in addition to the existing temporal components such as day of week and day of year, (2) to add line graphs along with other types of glyphs to the visual toolkit, (3) to enable an overlaying plot consisting of multiple time series. The proposed algorithm has been implemented as the `frame_calendar` function in the **sugrrants** package (Wang, Cook, and Hyndman 2017) using R (R Core Team 2017).

The remainder of the paper is organised as follows. Section 2 describes the `frame_calendar` function and the detailed construction. Section 3 presents some examples of its usage. Section 4 discusses the advantages and disadvantages of the method.

2. Construction

In the section, the algorithms to construct calendar grids, different scales, and reference lines and labels are discussed in detail.

```
frame_calendar(
  data, x, y, date, calendar = "monthly", dir = "h", sunday = FALSE,
  nrow = NULL, ncol = NULL, polar = FALSE, scale = "fixed"
)
```

2.1. Grid construction

The algorithm for constructing a calendar plot uses linear algebra, similar to that used in the glyph map displays for spatio-temporal data (Wickham, Hofmann, Wickham, and Cook 2012). To make a year long calendar, requires cells for days, embedded in blocks corresponding to months, organised into a grid layout for a year. Each month can be captured with 35 (5 $\times$ 7) cells, where the top left is Monday of week 1, and the bottom right is Sunday of week 5. These cells provide a micro canvas on which to plot the data. The first day of the month could be any of Monday-Sunday, which is determined by the year of the calendar. Months are

				k=5, g=5 i=1, j=5	*g=k+1 i=1, j=6*	*g=k+2 i=1, j=7*
g=k+3 i=2, j=1	*g=k+4 i=2, j=2*	*g=k+5 i=2, j=3*	*g=k+6 i=2, j=4*	*g=k+7 i=2, j=5*	*g=k+8 i=2, j=6*	*g=k+9 i=2, j=7*
g=k+10 i=3, j=1	*g=k+11 i=3, j=2*	*g=k+12 i=3, j=3*	*g=k+13 i=3, j=4*	*g=k+14 i=3, j=5*	*g=k+15 i=3, j=6*	*g=k+16 i=3, j=7*
g=k+17 i=4, j=1	*g=k+18 i=4, j=2*	*g=k+19 i=4, j=3*	*g=k+20 i=4, j=4*	*g=k+21 i=4, j=5*	*g=k+22 i=4, j=6*	*g=k+23 i=4, j=7*
g=k+24 i=5, j=1	*g=k+25 i=5, j=2*	*g=k+26 i=5, j=3*	*g=k+27 i=5, j=4*	...	...	*g=k+d i=5, j=7*

Figure 1: Illustration of the indexing layout for cells in a month.

of different length days, ranging form 28-31, and each month could extend over six weeks but the convention in these months is to wrap the last few days up to the top row of the block. The notation for creating these cells is as follows:

- $k = 1, \ldots, 7$ is the day of the week that is the first day of the month.
- $d = 28, 29, 30$ or 31 representing the number of days in any month.
- (i, j) is the grid position where $1 \leq i \leq 5$ is week within the month, $1 \leq j \leq 7$, is day of the week.
- $g = k, \ldots, (k + d)$ indexes the day in the month, inside the 35 possible cells.

The grid position for any day in the month is given by

$$\begin{aligned} i &= \lceil (g \mod 35)/7 \rceil, \\ j &= g \mod 7. \end{aligned} \tag{1}$$

Figure 1 illustrates this (i, j) layout for a month where $k = 5$.

To create the layout for a full year, (m, n) denotes the position of the month arranged in the plot, where $1 \leq m \leq M$ and $1 \leq n \leq N$. Between each month requires some small amount of white space, label this b. Figure 2 illustrates this layout.

Each cell forms a canvas on which to draw the data. Consider the canvas to have limits $[0, 1]$ horizontally and vertically. For the pedestrian sensor data, within each cell hour is plotted horizontally and count is plotted vertically. Each variable is scaled to have values between $[0, 1]$, using the minimum and maximum of all the data values to be displayed assuming of fixed scales. Let h be the scaled hour, and c the scaled count.

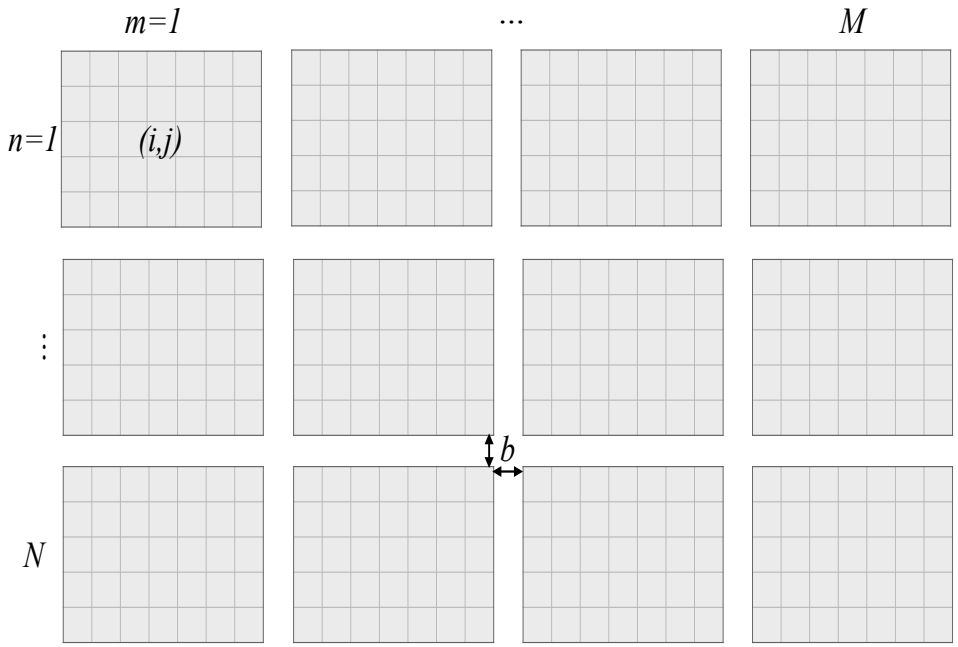

Figure 2: Illustration of the indexing layout for months of one year.

Then the final points for making the calendar line plots of the pedestrian sensor data is given by:

$$\begin{aligned} x &= i + (i-1) \times m + (m-1) \times b + h, \\ y &= -j - (j-1) \times n - (n-1) \times b + c. \end{aligned} \tag{2}$$

Note that for the vertical direction, the top left is the starting point of the grid, hence the subtractions, and resulting negative values to lay out the cells. Within each cell, the starting position is bottom left.

The algorithm can be relatively easily extended to layout from a single month to a few years, based on the period of time to be visualised. (M, N) can be determined by the user using the arguments of `nrow` and `ncol`. If the one would like to visualise data spanning over three years, for example, $M = 12$ and $N = 3$ seem to be an appropriate choice when assessing the differences of a given month across the years.

The algorithm can be simplified to accommodate the other two types of calendar formats. One is comprised of days of a week in columns and weeks of a year in rows, and the other is days of a month in columns and months of a year in rows, which we refer to as "weekly" and "daily" calendar respectively. Due to the arrangements, the weekly calendar puts more emphases on days of a week over days of a year, whereas the daily one serves as the opposite. The monthly format can be considered as the advanced twist of both weekly and daily. Temporal patterns lead to which format to be employed. The weekly calendar can be a nice attempt if the most variations can be characterised by days of a week. On the other hand, the absence of weekly effect but the presence of yearly effect can direct to the daily calendar. When both are present, the monthly calendar appears to be a better choice.

We only illustrate that grids are laid out over the horizontal direction in the paper. The

vertical direction can be enabled by swapping i and j in the algorithm stated above. It is particularly useful for those users who get accustomed to calendars of vertical organisation in some countries.

2.2. Scales

Section 2.1 discusses the implementation applied to the fixed scale which is using the whole range of all the data values. In addition to the fixed scale (`fixed`), the remaining options include free scale on all the days (`free`), day of the week only (`free_wday`), and day of the month only (`free_mday`) for different comparisons.

Grouping the cells according to the common period give rise to these scales. The minimums and maximums obtained for every (i, j) and (m, n) allow for individual scale over all the single days. Similarly, j indexing day of the week is bundled to determine the scales on days of a week, and g representing the day in the month is for the scales on days of a month.

All the figures shown before are drawn on the fixed scale so that it makes the magnitudes comparable over the whole period of time. On the other hand, some sub-series of small magnitudes yet worth-noting shapes possibly become invisible whilst embedded in the overall picture. These sorts of scales, as a result, complements the calendar-based visualisation in general. (NEED FIGURES)

2.3. Reference lines and labels

Reference lines dividing each cell and block as well as labels indicating weekday and month are also provided in order to make calendar-based graphics more accessible and informative.

Regarding the monthly calendar, the major reference lines separate every month panel and the minor ones separate every cell, represented by the think and thin lines respectively. The major reference lines are placed surrounding every month block: for each m, the vertical lines are determined by $\min(x)$ and $\max(x)$; for each n, the horizontal lines are given by $\min(y)$ and $\max(y)$. The minor reference lines are placed on the left side of every cell: for each i, the vertical division is $\min(x)$; for each j, the horizontal is $\min(y)$.

The abbreviated month labels located on the top left are obtained through $(\min(x), \max(y))$ for every m and n. The weekday texts with a single letter are uniformly placed on the bottom of the whole canvas, that is $\min(y)$, with the central position of a cell $x/2$ for each j.

3. Examples

4. Discussion

The calendar-based visualisation provides data plots in the familiar (at least for the Western world) format of an everyday tool. Special events for the region, like Anzac Day in Australia, or Thanksgiving Day in the USA, more easily pop out to the viewer as public holidays, rather than a typical work day.

This sort of layout may be useful for studying consumer trends, or human behaviour, like the pedestrian patterns. It may not work so well for physical patterns like temperature, which

are not typically affected by human activity.

The limitation is also evident: hard to perceive trend as not on the common scale.

References

City of Melbourne (2017). *Pedestrian Volumn in Melbourne*. Town Hall, 90-120 Swanston Street, Melbourne VIC 3000. URL http://www.pedestrian.melbourne.vic.gov.au.

Jacobs J (2017). *ggcal: Calendar Plot Using ggplot2*. R package version 0.1.0.

Kothari A, Ather (2016). *ggTimeSeries: Nicer Time Series Visualisations with ggplot syntax*. R package version 0.1, URL https://github.com/Ather-Energy/ggTimeSeries.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Van Wijk JJ, Van Selow ER (1999). "Cluster and Calendar Based Visualization of Time Series Data." In *Information Visualization, 1999.(Info Vis' 99) Proceedings. 1999 IEEE Symposium on*, pp. 4–9. IEEE.

Wang E, Cook D, Hyndman R (2017). *sugrrants: Supporting Graphics with R for Analysing Time Series*. R package version 0.0.1.9000, URL http://pkg.earo.me/sugrrants.

Wickham H, Hofmann H, Wickham C, Cook D (2012). "Glyph-maps for Visually Exploring Temporal Patterns in Climate Data and Models." *Environmetrics*, **23**(5), 382–393.

Affiliation:

Earo Wang
Monash University
Department of Econometrics and Business Statistics, Monash University, VIC 3800 Australia
E-mail: earo.wang@monash.edu

Di Cook
Monash Univeristy
Department of Econometrics and Business Statistics, Monash University, VIC 3800 Australia
E-mail: dicook@monash.edu

Rob J Hyndman
Monash Univeristy
Department of Econometrics and Business Statistics, Monash University, VIC 3800 Australia
E-mail: rob.hyndman@monash.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd