*Journal of Statistical Software*

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

# Calendar-based graphics for visualizing people's daily schedules

**Earo Wang**
Monash University

**Dianne Cook**
Monash University

**Rob J Hyndman**
Monash University

**Abstract**

This paper describes a `frame_calendar` function that organizes and displays temporal data, collected on sub-daily resolution, into a calendar layout. Calendars are broadly used in society to display temporal information, and events. The `frame_calendar` uses linear algebra on the date variable to create the layout. It utilizes the grammar of graphics to create the plots inside each cell, and thus synchronises neatly with **ggplot2** graphics. The motivating application is studying pedestrian behavior in Melbourne, Australia, based on counts which are captured at hourly intervals by sensors scattered around the city. Faceting by the usual features such as day and month, was insufficient to examine the behavior. Making displays on a monthly calendar format helps to understand pedestrian patterns relative to events such as work days, weekends, holidays, and special events. The layout algorithm has several format options and variations. It is implemented in the R package **sugrrants**.

*Keywords*: data visualization, statistical graphics, time series, R package, grammar of graphics.

## 1. Introduction

We develop a method for organizing and visualizing temporal data, collected at sub-daily intervals, into a calendar layout. The purpose of the calendar-based visualization is to provide insights into people's daily schedules relative to events such as work days, weekends, holidays, and special events. This work was originally motivated by studying foot traffic in the city of Melbourne, Australia (City of Melbourne 2017). There have been 43 sensors installed that count pedestrians every hour across the downtown area (Figure 1). The dataset can shed light on people's daily rhythms, and assist the city administration and local businesses with event planning and operational management. A routine examination of the data would involve constructing conventional time series plots to catch a glimpse of temporal patterns.

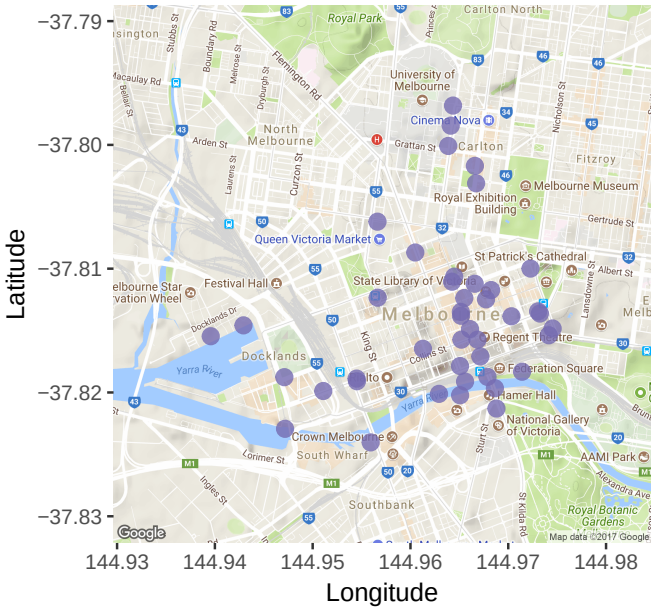


Figure 1: Map of the Melbourne city area with purple dots indicating sensor locations.

The faceted plots in Figure 2, give an overall picture of the foot traffic at 3 different sensors over 2016. Further faceting by day of the week (Figure 3) provides a better glimpse of the daily and sub-daily pedestrian patterns. The sensor data, like many temporal datasets on human behaviors, lends itself to a number of exploratory data visualization challenges:

1. Variations exist on multiple time scales including time of day, day of week, and day of year (such as public holiday and recurring events).
2. Since the data are often collected at sub-daily frequencies, they typically involve a large number of observations.
3. Measurements of a single type are made at multiple locations at a given time point, which creates the need for comparing and contrasting between locations.

The conventional ways to display time series and constructing faceted displays are insufficient to address these challenges.

The work is inspired by Wickham, Hofmann, Wickham, and Cook (2012), which uses linear algebra to display spatio-temporal data as glyphs on maps. It is also related to recent work by Hafen (2017) which provides methods in the **geofacet** package to arrange data plots into a grid, while preserving the geographical position. Both of these show data in a spatial context.

In contrast, calendar-based graphics unpacks the temporal variable, at different resolutions, to digest multiple seasonalities, and special events. There is some existing work in this area. For example, Van Wijk and Van Selow (1999) developed a calendar view of the heatmap to represent the number of employees in the work place over a year, where colours indicate different clusters derived from the days. It contrasts week days and weekends, highlights public holidays, and presents other known seasonal variation such as school vacations, all of which have influence over the turn-outs in the office. The calendar-based heatmap was implemented in two R packages: **ggTimeSeries** (Kothari and Ather 2016) and **ggcal** (Jacobs

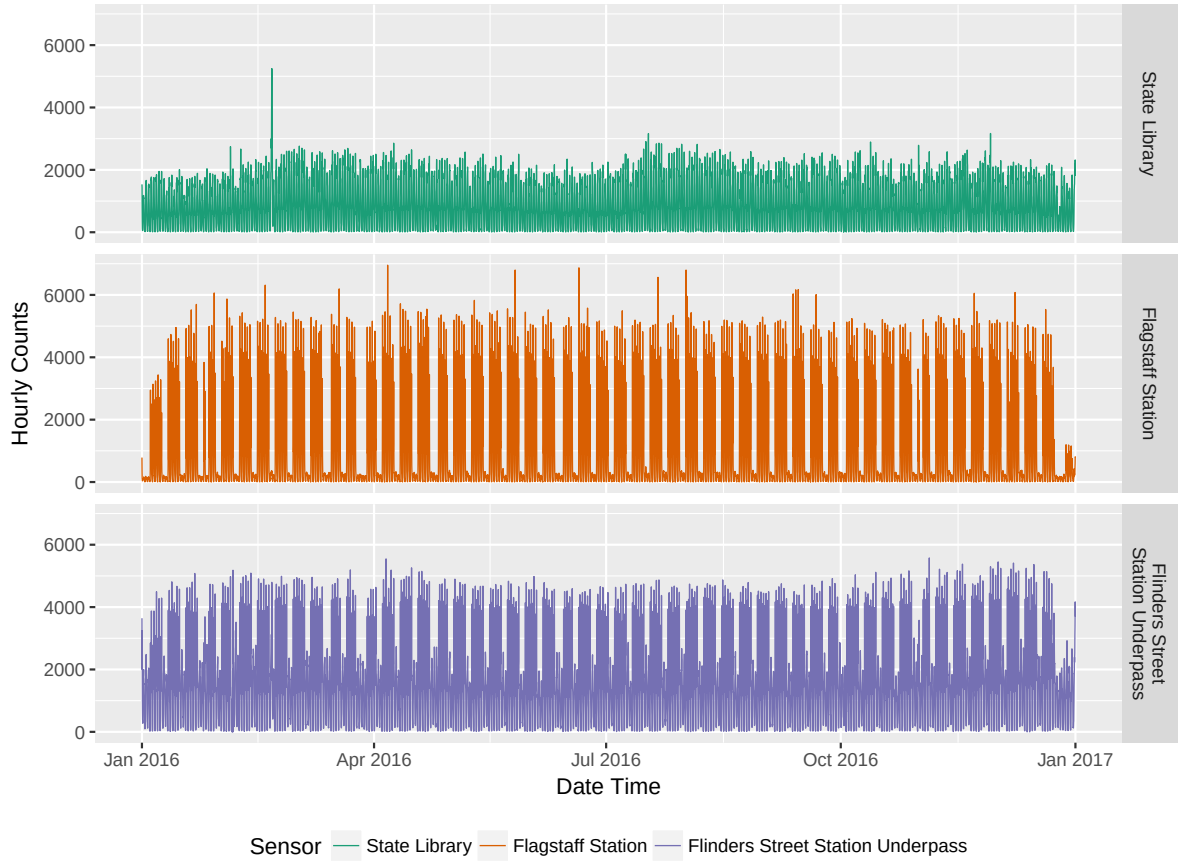


Figure 2: Time series plot showing the number of pedestrians in 2016 measured at 3 different sensors in the city of Melbourne. Coloured by the sensors, small multiples of lines show that the foot traffic varies from one sensor to another in terms of both time and number. The weekly patterns look distinctive across these 3 sensors. There is an eye-catching spike which occurred at the State Library, caused by the annual White Night event on 20th of February.

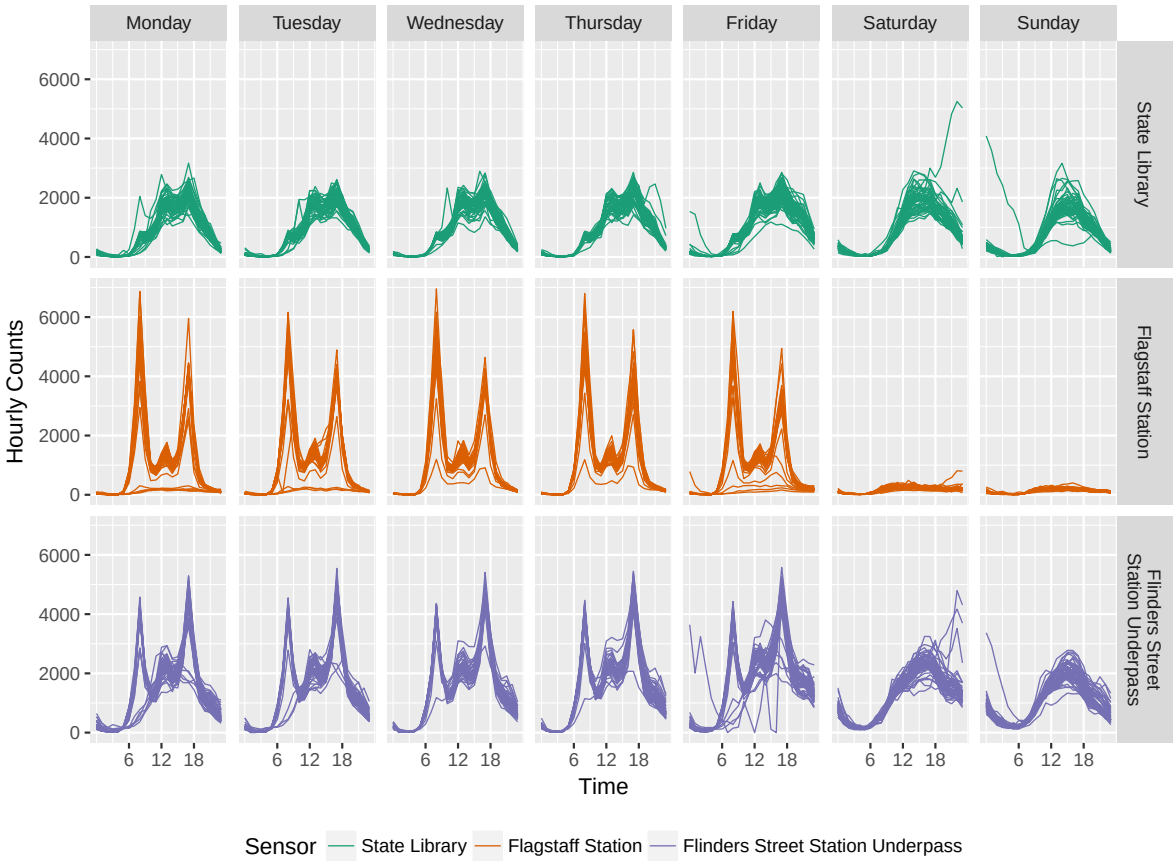


Figure 3: Hourly pedestrian counts faceted by sensors and days of the week using lines. It features at least two types of seasons—time of day and day of week—across all the sensors.

2017). However, these techniques are limited to colour-encoding graphics and are unable to use time scales smaller than a day. Time of day, which serves as one of the most important aspects in explaining substantial variations arising from pedestrian sensor data, will be neglected through daily aggregation. Additionally, if simply using colored blocks rather than curves, it may become perceptually difficult to estimate the shape positions and changes, although using curves comes with the cost of more display capacity (Cleveland and McGill 1984; Lam, Munzner, and Kincaid 2007).

We propose a new algorithm to go beyond the calendar-based heatmap. The approach is developed with three conditions in mind: (1) to display time-of-day variation in addition to longer temporal components such as day-of-week and day-of-year; (2) to incorporate line graphs and other types of glyphs into the graphical toolkit for the calendar layout; (3) to enable overlaying plots consisting of multiple time series. The proposed algorithm has been implemented in 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 organized as follows. Section 2 demonstrates the construction of the calendar layout in depth. Section 3 lists and describes the options that come with the `frame_calendar` function. Section 4 presents some variations of its usage. Section 5 discusses the advantages and disadvantages of the method.

## 2. Construction

Figure 4 shows the line glyphs framed in the monthly calendar over the year 2016. This is achieved by the `frame_calendar` function computing the new coordinates according to the input data variables; in turn the rearranged data values are plotted using the **ggplot2** package (Wickham and Chang 2016), which is an implementation of the grammar of graphics (Wilkinson 2005; Wickham 2010).

The algorithm for constructing a calendar plot uses linear algebra, similar to that used in the glyph map displays for spatio-temporal data (Wickham *et al.* 2012). To make a year long calendar requires cells for days, embedded in blocks corresponding to months, organized 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 by default. 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 of different lengths, ranging from 28 to 31 days, 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.

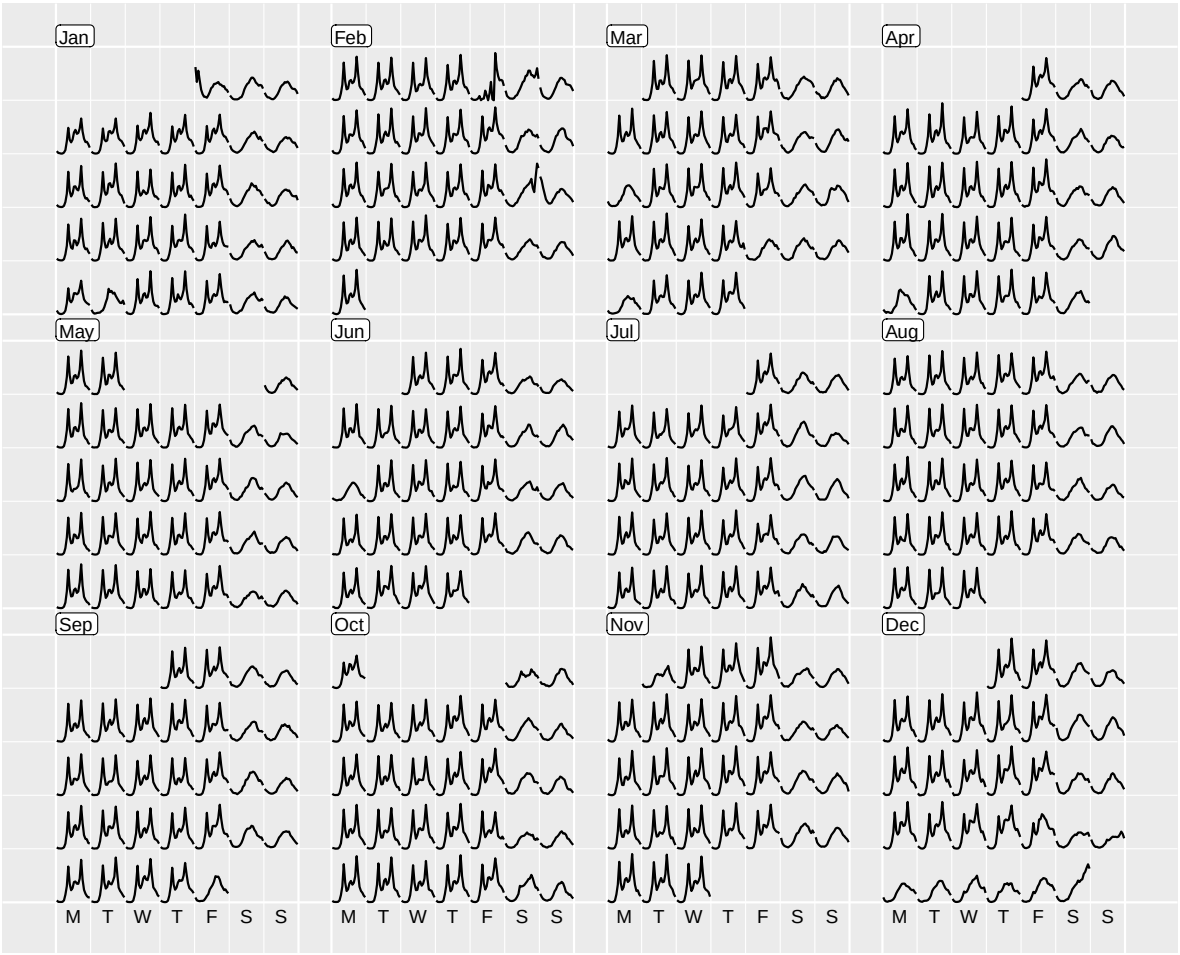


Figure 4: The calendar-based display of hourly foot traffic at Flinders Street Station using line glyphs. The arrangement of the data into a $3 \times 4$ monthly grid represents all the traffic in 2016. The disparities between week day and weekend along with public holiday are immediately apparent.

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

Figure 5: Illustration of the indexing layout for cells in a month.

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 5 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, denoted by $b$. Figure 6 illustrates this layout where $M = 3$ and $N = 4$.

Each cell forms a canvas on which to draw the data. Initialize the canvas to have limits $[0, 1]$ both 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 in $[0, 1]$, using the minimum and maximum of all the data values to be displayed, assuming fixed scales. Let $h$ be the scaled hour, and $c$ the scaled count.

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

$$\begin{aligned} x &= j + (n-1) \times 7 + (n-1) \times b + h, \\ y &= i - (m-1) \times 5 - (m-1) \times b + c. \end{aligned} \tag{2}$$

Note that for the vertical direction, the top left is the starting point of the grid (in Figure 5) which is why subtraction is performed. Within each cell, the starting position is the bottom left.

In order to make calendar-based graphics more accessible and informative, reference lines

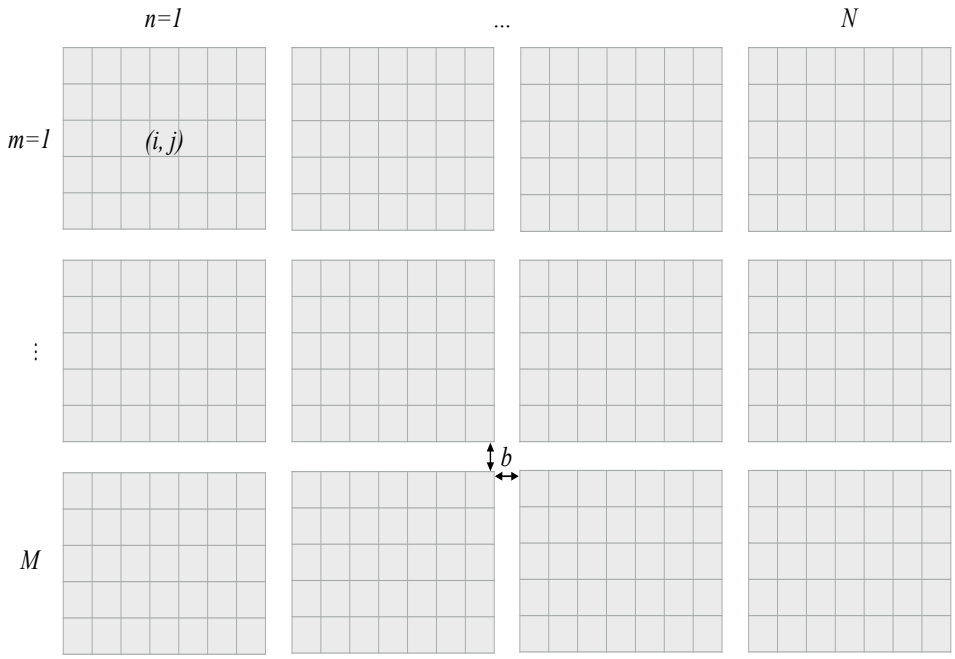


Figure 6: Illustration of the indexing layout for months of one year.

dividing each cell and block as well as labels indicating week day and month are also computed before plotting construction.

Regarding the monthly calendar, the major reference lines separate every month panel and the minor ones separate every cell, represented by the thick and thin lines in Figure 4, 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 only 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 month labels located on the top left are obtained through $(\min(x), \max(y))$ for every $(m, n)$. The week day texts are uniformly positioned on the bottom of the whole canvas, that is $\min(y)$, with the central position of a cell $x/2$ for each $j$.

## 3. Options

There are several options provided for the `frame_calendar` function to initialize and adjust the display of a calendar plot:

```
frame_calendar(
  data, x, y, date, calendar = "monthly", dir = "h", sunday = FALSE,
  nrow = NULL, ncol = NULL, polar = FALSE, scale = "fixed",
  width = 0.95, height = 0.95
)
```

Assuming that tidy data (Wickham 2014) is the underlying data structure, the parameter `x` takes a variable that will be mapped to the horizontal axis and the parameter `y` mapped

to the vertical axis for plot construction. In Figure 4, for example, the `x` is the variable specifying the time of the day, and the `y` is the variable representing the hourly counts. The `date` argument is given by the date variable that determines the correct order of the calendar layout.

The algorithm can be extended to display data from a single month up to a few years. The number of rows and columns in the layout can be specified with the arguments `nrow` and `ncol`. If the one would like to visualize data spanning over three years, for example, `nrow = 3` and `ncol = 12` would be an appropriate choice when assessing the differences of a given month across the years.

In Section 2 we only illustrated that grids are laid out horizontally. The vertical direction can be enabled by swapping $i$ and $j$ in equation (1), which occurs when the argument `dir` is set to `"v"`. This benefits users who are accustomed to calendars that are organized vertically, which is common in some countries. Next we shall describe some of the arguments that allow for alternate displays.

### 3.1. Layouts

The algorithm described in Section 2 is for the most common calendar layout, the "monthly" calendar, but it can be simplified to accommodate two other types 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 the "weekly" and "daily" calendar, respectively. The layout to be used is controlled by the `calendar` argument. The weekly calendar puts more emphasis on days of a week over days of a year, whereas the daily calendar does the opposite. The monthly calendar can be considered as the advanced variant of both weekly and daily. Temporal patterns motivate which variant should be employed. The weekly calendar is appropriate if most of the variation can be characterised by days of the week. On the other hand, the daily calendar should be used when there is a yearly effect but not a weekly effect in the data (for example weather data). When both effects are present, the monthly calendar would be a better choice.

### 3.2. Polar transformation

When `polar = TRUE`, a polar transformation is carried out on the data. The computation is similar to the one described in Wickham *et al.* (2012). Figure 7 shows star plots embedded in the monthly calendar layout, which is equivalent to Figure 4 placed in polar coordinates.

### 3.3. Scales

Section 2 discusses the implementation applied to a fixed scale, which uses the range over all data values. The `scale` argument controls the scaling of the display. The fixed scale (`fixed`) is the default, meaning that the data values in all positions are scaled. The remaining options include: free scale within each cell (`free`), cells derived from the same day of the week (`free_wday`), or cells from the same day of the month (`free_mday`). The scaling allows for the comparisons of absolute or relative values, and the emphasis of different temporal variations.

Grouping the cells based on the corresponding time period gives rise to the different scales. For example, the minimums and maximums obtained from each cell produce a local scale. The overall variation gives way to the individual shape. Figure 8 is an example of plotting line

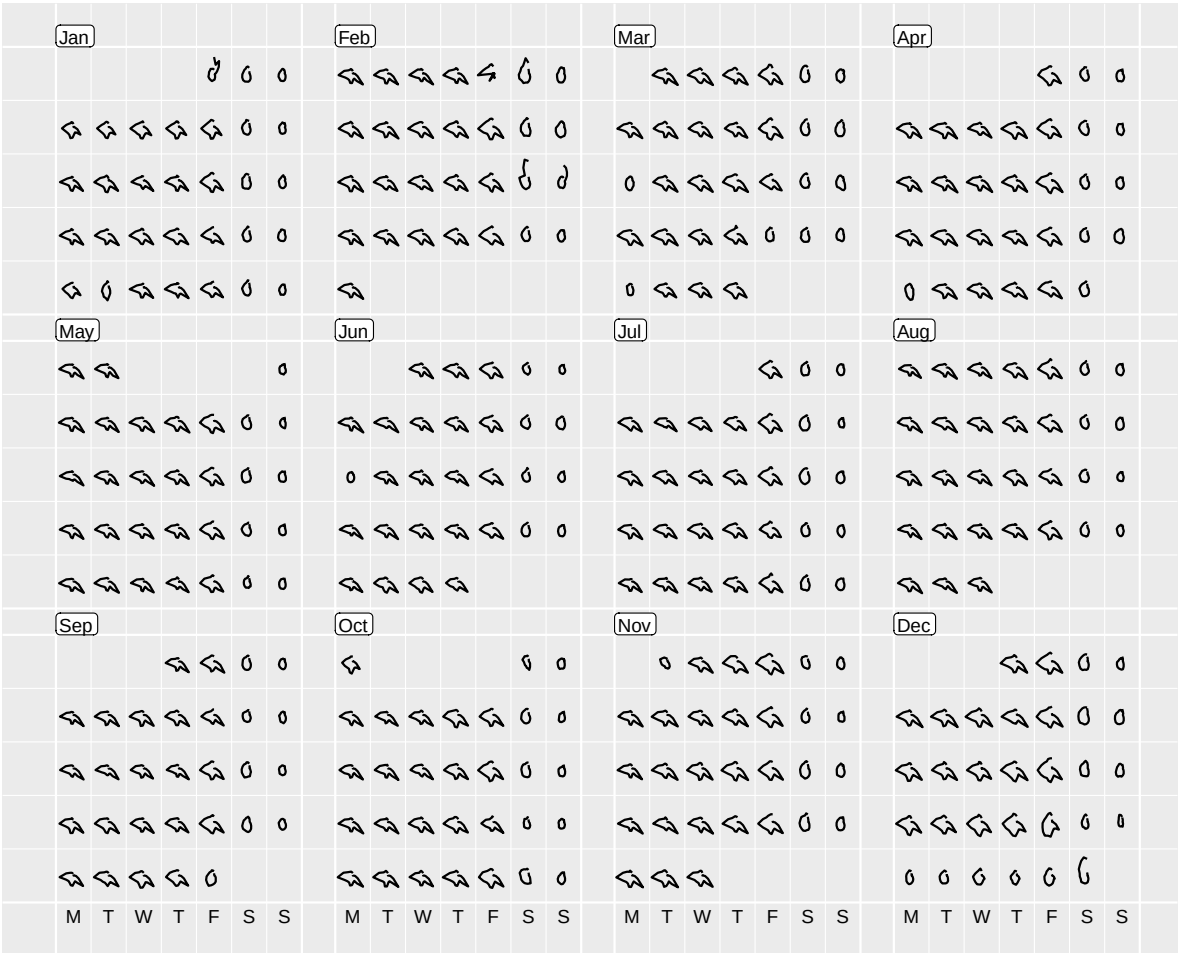


Figure 7: Figure 4 in circular layout, which is referred to as star plots. The daily periodicity on work days are clearly visible.

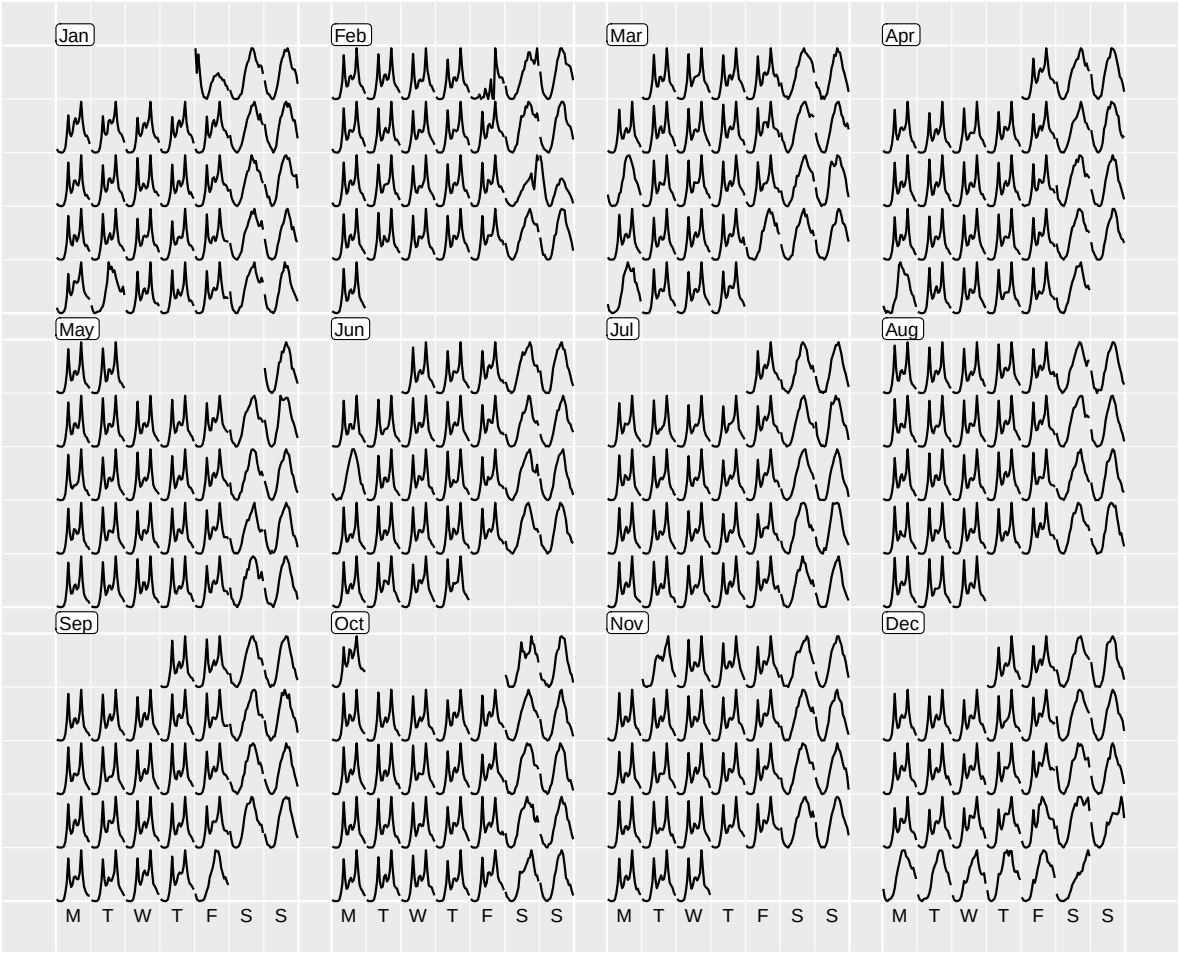


Figure 8: Line glyphs on the calendar format showing hourly foot traffic at Flinders Street Station, scaled over all the days. The individual shape on a single day becomes more distinctive, however it is impossible to compare the size of peaks between days.

charts scaled locally using `scale = "free"`. The daily variation is more distinctive compared to Figure 4.

The `free_wday` suggests that the scales are common for a given week day across all weeks, but different for each week day within a week. In other words, the same value within a given Monday, for example, corresponds to the same position across all Mondays but a different position in other week days. Similarly, the `free_mday` suggests free scaling for any day within a given month. The scaling choices combined with the calendar layouts offers a number of varieties to construct the plot.

# 4. Variations

## 4.1. Overlaying and faceting subsets

The comparison of one sensor to another adds additional insights to the dataset, which is

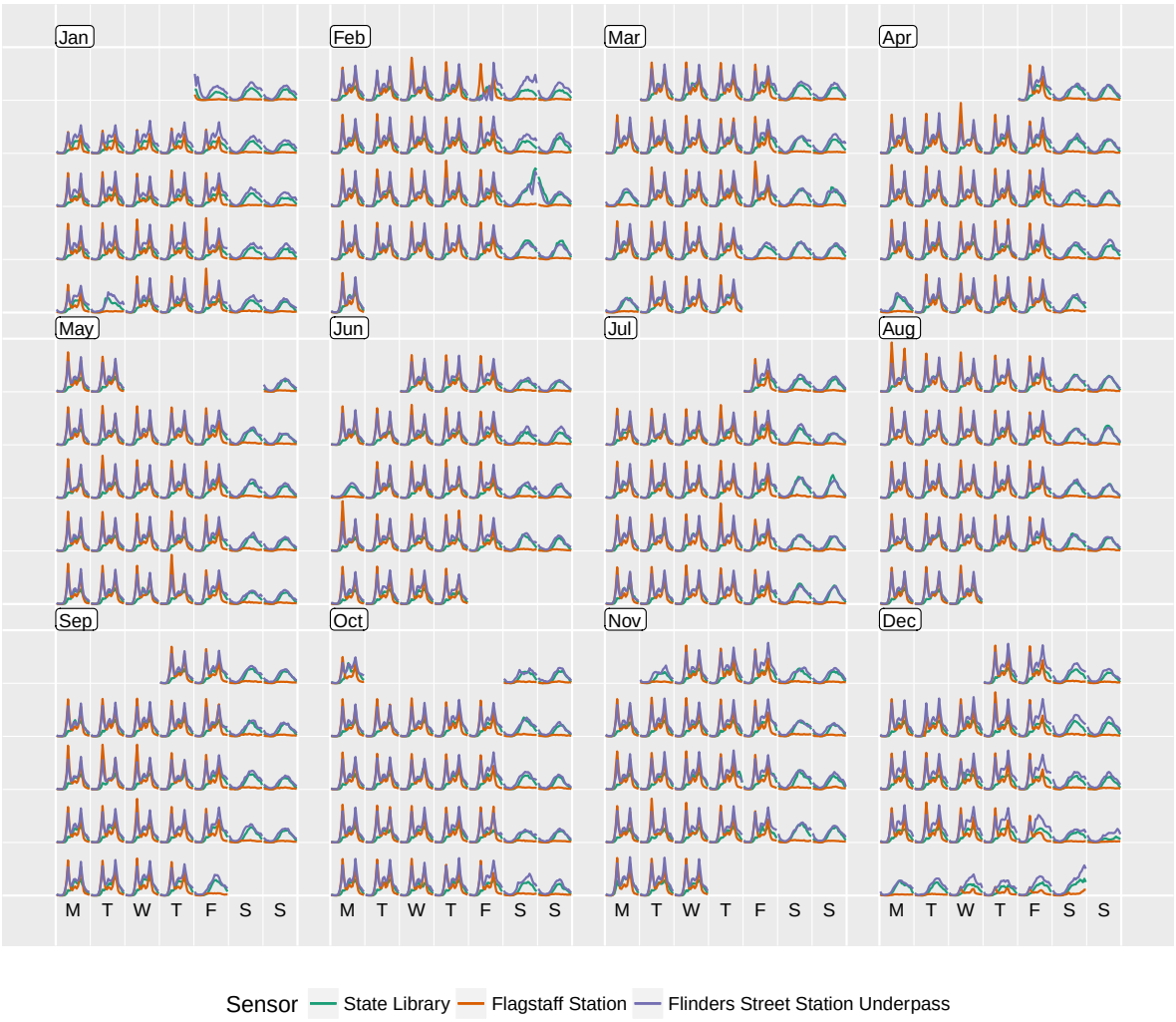


Figure 9: Overlaying line graphs of the 3 sensors in the monthly calendar. Flagstaff station is not as busy as the other two on non-work days.

often done with an overlaying plot such as Figure 9. For instance, the dominant patterns that occurred at Flinders Street and Flagstaff train stations are driven by the commuters on work days; however, Flinders Street Station has higher pedestrian counts during the weekends and public holidays. This suggests that Flagstaff Station limits its functionality on non-work days, but various activities take place around the Flinders Street Station other than commuting. Figure 9 also shows that the State Library follows a similar temporal trend as Flinders Street Station on non-work days. The nighttime events, such as White Night and New Year's Eve, have barely affected the operation of Flagstaff Station but heavily affected the incoming and outgoing traffic to Flinders Street Station and the State Library.

To avoid the overlapping problem, the calendar layout can be embedded into a series of subplots for the different sensors. Figure 10 presents the idea of faceting calendar plots. This allows comparing the overall structure between sensors, while emphasising individual sensor variation.

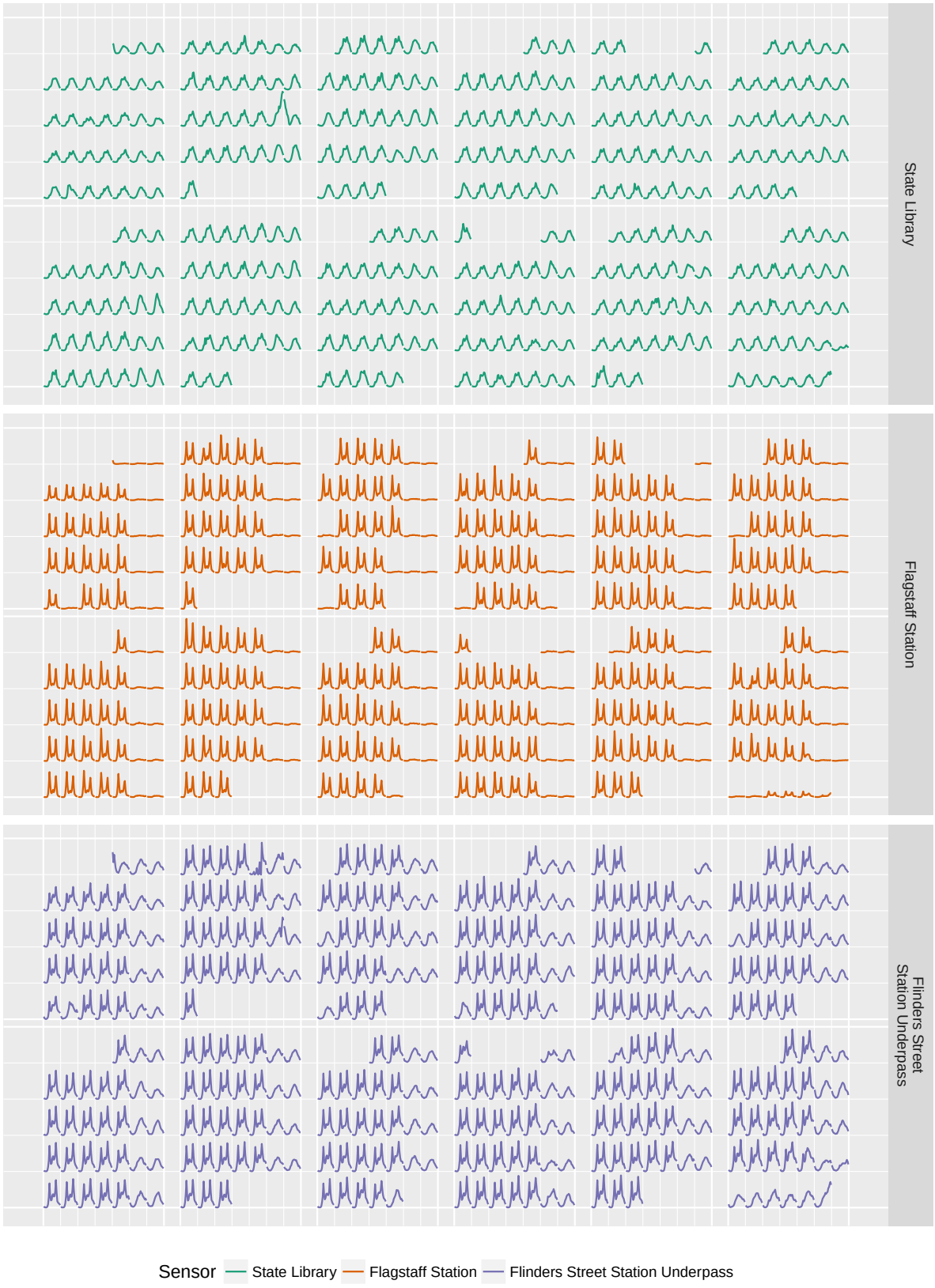


Figure 10: Line charts, embedded in the $6 \times 2$ monthly calendar, coloured and faceted by the 3 sensors. The variations of an individual sensor are emphasised, and the shapes can be compared across the cells and sensors.

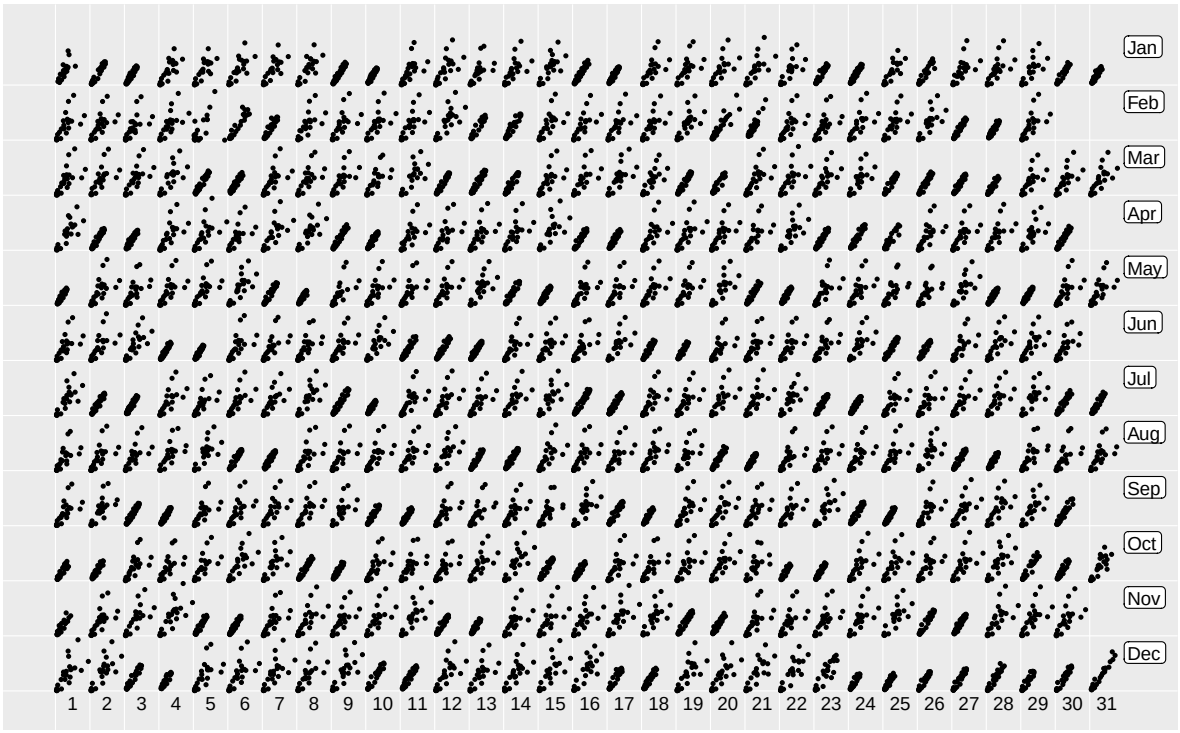


Figure 11: Lag scatterplot in the daily calendar layout. Previous hour's count is plotted against each hour's count at Flinders Street Station to demonstrate the autocorrelation at lag 1. The correlation between them is more consistent on non-work days than work days.

### 4.2. Different types of plots

The `frame_calendar` function is not constrained to mapping only the temporal variable to the `x` argument. Figure 11 shows a lag scatterplot at Flinders Street Station, where the current hourly count is assigned to the `x` argument and the lagged hourly count to the `y` argument. This figure is organized in the daily calendar layout. It provides a visual tool for identifying repeating patterns. Figure 11 indicates two separate paths in the work-day glyphs, suggesting that an hour of a work day with many pedestrians is likely to be followed in the next hour with either greater or fewer numbers of pedestrians. However, this pattern is absent on non-work days. There is a bimodality on work days while there is unimodality on the rest of the days, which is also supported by Figure 4.

The newly computed coordinates can also work with more complicated plots, such as boxplots. Figure 12 uses a loess smooth line superimposed on side-by-side boxplots. It shows the distribution of hourly counts across all 43 sensors during December. The last week of December is the holiday season: people are off work on the day before Christmas, go shopping on the Boxing day, and stay out for the fireworks on New Year's Eve.

We also offer languages other than English for text labelling. Figure 13 shows the same plot as Figure 12 labelled using simplified Chinese characters.

## 5. Discussion

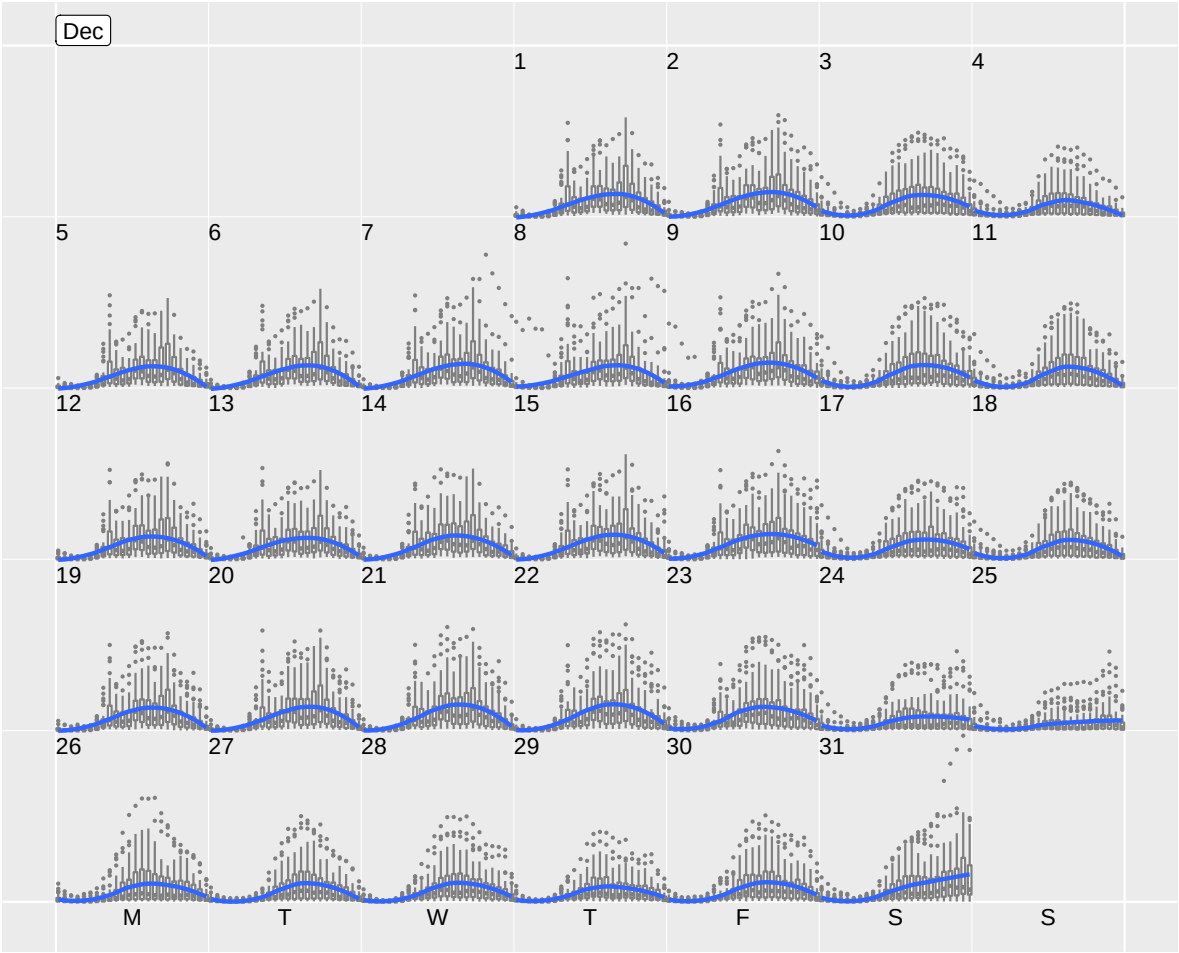


Figure 12: Side-by-side boxplot of hourly counts across all the 43 sensors in December of 2016, with the superimposing loess smooth line for each day. It shows the hourly distribution in the city as a whole. There is one sensor attracting a larger number of people on New Year's Eve than the rest.

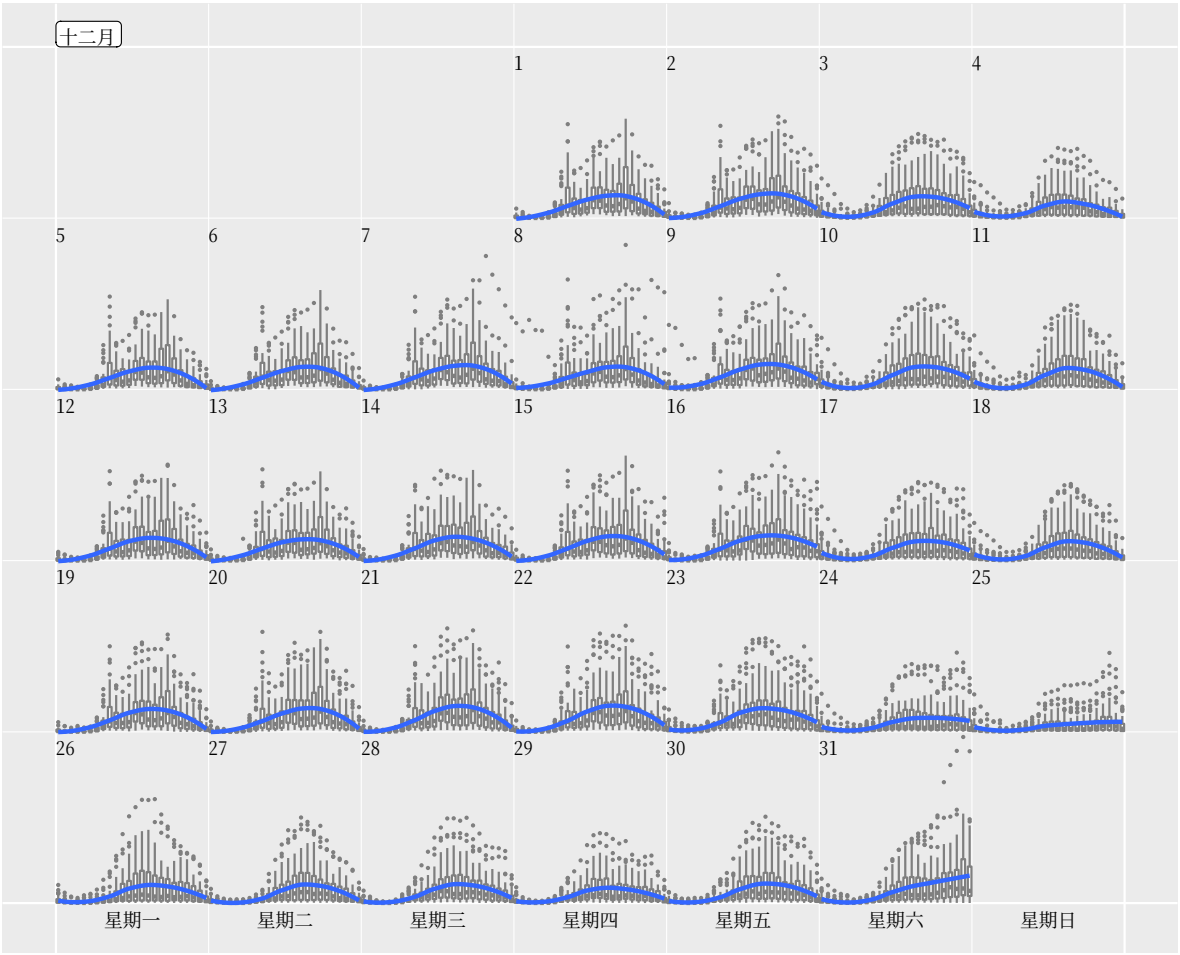


Figure 13: The same side-by-side boxplot as Figure 12, but with the month and week day labels in Chinese. It demonstrates the natural support for languages other than English.

The calendar-based visualization provides data plots in the familiar format of an everyday tool. Patterns on 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, than they would on a more commonly used week day and month facetted layout.

This sort of layout will be useful for studying consumer trends, or human behavior, such as pedestrian patterns or pollution peaks. It will not be so useful for physical patterns like temperature, which are not typically affected by human activity. The layout does not replace traditional displays, but serves to complement to further tease out structure in temporal data. Analysts would still be advised to plot overall summaries and deviations, in order to study general trends.

The layout is achieved by utilizing linear combinations of temporal components and measured variables. This provides the data with the most screen real estate. It could be useful to develop this into a fully-fledged facetting method, with formal labels and axes. This is a future goal.

## References

City of Melbourne (2017). *Pedestrian Volume in Melbourne*. URL http://www.pedestrian.melbourne.vic.gov.au.

Cleveland WS, McGill R (1984). "Graphical perception: Theory, experimentation, and application to the development of graphical methods." *Journal of the American Statistical Association*, **79**(387), 531–554.

Hafen R (2017). *geofacet: 'ggplot2' Faceting Utilities for Geographical Data*. R package version 0.1.4, URL https://CRAN.R-project.org/package=geofacet.

Jacobs J (2017). *ggcal: Calendar Plot Using ggplot2*. R package version 0.1.0, URL https://github.com/jayjacobs/ggcal.

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

Lam H, Munzner T, Kincaid R (2007). "Overview Use in Multiple Visual Information Resolution Interfaces." *IEEE Transactions on Visualization and Computer Graphics*, **13**(6), 1278–1285.

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 RJ (2017). *sugrrants: Supporting Graphs for Analysing Time Series*. R package version 0.1.0, URL https://CRAN.R-project.org/package=sugrrants.

Wickham H (2010). "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics*, **19**(1), 3–28.

Wickham H (2014). "Tidy Data." *Journal of Statistical Software*, **59**(10), 1–23.

Wickham H, Chang W (2016). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 2.2.1, URL https://CRAN.R-project.org/package=ggplot2.

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.

Wilkinson L (2005). *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

## Affiliation:

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

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

Rob J Hyndman
Monash University
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