

# **Tidy tools for supporting fluent workflow in temporal data analysis**

A thesis submitted for the degree of

Doctor of Philosophy

by

Earo Wang

B.Comm. (Hons), Monash University



Department of Econometrics and Business Statistics

Monash University

Australia

September 2019

# Contents

<b>Acknowledgements</b>	<b>v</b>
<b>Declaration</b>	<b>vii</b>
<b>Preface</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Calendar-based graphics . . . . .	2
1.2 Tidy temporal data structure . . . . .	2
1.3 Missingness in time . . . . .	2
1.4 Summary . . . . .	3
<b>2 Calendar-based graphics for visualizing people's daily schedules</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 Creating a calendar display . . . . .	10
2.3 Case study . . . . .	25
2.4 Discussion . . . . .	30
Acknowledgements . . . . .	30
<b>3 A new tidy data structure to support exploration and modeling of temporal data</b>	<b>31</b>
3.1 Introduction . . . . .	32
3.2 Data structures . . . . .	33
3.3 Contextual semantics . . . . .	36
3.4 Temporal data pipelines . . . . .	40
3.5 Software structure and design decisions . . . . .	44
3.6 Case studies . . . . .	49
3.7 Smart-grid customer data in Australia . . . . .	53
3.8 Conclusion and future work . . . . .	59
Acknowledgments . . . . .	59
<b>4 Data representation, visual and analytical techniques for demystifying temporal missing data</b>	<b>61</b>
4.1 Introduction . . . . .	61
<b>5 Conclusion and future plans</b>	<b>63</b>

## CONTENTS

---

5.1 Software development . . . . .	63
5.2 Future work . . . . .	64
<b>Bibliography</b>	<b>67</b>



# Acknowledgements

First and foremost, I would like to thank my supervisors, [Dianne Cook](#) and [Rob J Hyndman](#).



# **Declaration**

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma in any university or equivalent institution, and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Earo Wang



# Preface

Chapter 2 has been tentatively accepted by the *Journal of Computational and Graphical Statistics* for the second-round revision. It has won the ASA Statistical Graphics Student Paper Award and the ACEMS Business Analytics Prize in 2018. Chapter 3 has been tentatively accepted and resubmitted to the *Journal of Computational and Graphical Statistics*. The accompanying R package **tsibble** has won the John Chambers Statistical Software Award in 2019. Chapter 4 is under development.

## Open and reproducible research

This thesis is written in R Markdown (Xie, Allaire, and Grolemund, 2018) with **bookdown** (Xie, 2016). The online version of this thesis is hosted at <https://thesis.earo.me>, powered by **Netlify**. All materials (including the data sets and source files) required to reproduce this document can be found at the Github repository <https://github.com/earowang/thesis>.

## License

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#).

The code used in this document is available under the [MIT license](#).



# Chapter 1

## Introduction

This thesis makes three original contributions for the analysis of temporal data. All three are grounded in an exploratory data analysis pipeline for time-indexed data. The research begins with a new technique for visualising data using a calendar layout, that fits neatly into a pipeline workflow. It is most useful when the data relates to human activity, and embeds plots in the familiar calendar. The second contribution is a new data abstraction which streamlines transformation, visualisation, and modelling for temporal data analysis. This “tsibble” data object is infrastructure holding the foundation of time series pipelines. The tsibble representation exposes a need for conceptually framing the handling of temporal missing values in a data-centric workflow. This is the third contribution of the thesis: exploratory and explanatory tools for understanding missing patterns in time.

Time series analysis has assumed that the entry point to data analysis is at model-ready data format, which provides little organisation or conceptual oversight on how one should get the wild data into a tamed state. This mind-set is related to a long-held belief that exploratory data analysis is a highly ad hoc statistical area, impossible to teach or formalise. However, the **tidyverse** framework, as originating in Wickham (2014) fundamentally overturns this thinking. Data plots and data wrangling, for which the “tidy data” conceptualisation supports, can be formally described using an abstract grammar. The grammar of graphics and data manipulation, as implemented in the **ggplot2** (Wickham et al., 2019a)

and **dplyr** (Wickham et al., 2019b) R packages respectively, form the core of the **tidyverse** suite of tools. My contributions extend the **tidyverse** way of thinking to the temporal domain, by providing tidy tools for supporting fluent workflow in temporal data analysis.

## 1.1 Calendar-based graphics

Visualisation is critical for understanding the data patterns, and for discovering the unexpected in data. Chapter 2 describes how to make a calendar layout for plots, so that human activity data can be examined in a familiar form. I develop a new method for assembling small multiples into a calendar layout. It is fully integrated into the grammar of graphics (Wilkinson, 2005; Wickham, 2009), specifying the plots as a functional mapping from data variables to graphical primitives. This new calendar-based display sketches the patterns of daily and sub-daily human behaviours, unlocking vivid and detailed data stories about the way we live. The chapter contains a comprehensive literature review related to calendar-based graphics.

## 1.2 Tidy temporal data structure

Data representation is an important component of data science research. “Tidy data” (Wickham, 2014) is the fundamental data architecture of the **tidyverse** ecosystem, making it possible to build fluent data pipelines for transformation, visualisation, and modelling. It does not adequately describe temporal data, and the current time series structures that are useful for modelling, inhibits a data-centric workflow. This research develops a new tidy data abstraction for temporal data, which lubricates the plumbing of temporal data analysis. Chapter 3 describes this abstraction, the process of using it for time series analysis, and a comprehensive literature review of related work.

## 1.3 Missingness in time

Chapter 4 develops exploratory and explanatory tools for understanding missing patterns in time, that fits into a tidy data workflow. This is a work in progress.

## 1.4 Summary

Chapter 2 provides full details of the calendar plot, algorithm and application. Chapter 3 explains the new data abstraction—tsibble—and illustrates how it can be used to form the basis of exploratory methods, visualisation and modelling of temporal data. Chapter 4 outlines a new nomenclature for temporal missing patterns, tools to identify these processes in data, and evaluate imputation results.

Chapter 5 summarises the software tools developed for the work and the impact, and discusses the future plans.



## **Chapter 2**

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

Calendars are broadly used in society to display temporal information and events. This paper describes a new calendar display for plotting data, that includes a layout algorithm with many options, and faceting functionality. The functions use modular algebra on the date variable to restructure the data into a calendar format. The user can apply the grammar of graphics to create plots inside each calendar cell, and thus the displays synchronize 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, is 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 functions for the calendar algorithm are available in the R package **sugrrants**

### **2.1 Introduction**

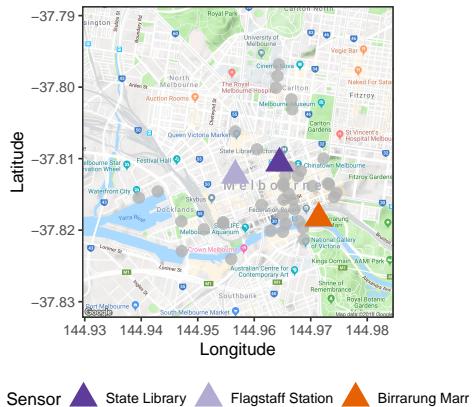
A new method for organizing and visualizing temporal data, collected at sub-daily intervals, into a calendar layout is developed. The format is created using modular arithmetic, giving a restructuring of the data, that can then be integrated into a data pipeline. The core

component of the pipeline is to visualize the resulting data using the grammar of graphics (Wilkinson, 2005; Wickham, 2009), as used in **ggplot2** (Wickham et al., 2019a), where plots are defined as a functional mapping from variables in the data to graphical elements. The data restructuring approach is consistent with the tidy data principles available in the **tidyverse** suite of tools (Wickham, 2017). The methods are implemented in a new R package called **sugrrants** (Wang, Cook, and Hyndman, 2019a).

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 are many sensors installed across the inner-city area, that count pedestrians every hour (Figure 2.1). Data from 43 sensors in 2016 is analyzed here. This data can shed light on people's daily rhythms, and assist the city administration and local businesses with event planning and operational management. Patterns relative to special events (such as public holidays and recurring cultural/sporting events) would be worth studying in comparison to regular days, but the conventional displays of time series data may bury this detail.

A routine examination of the data would involve constructing a time series plot to examine the temporal patterns. The faceted plots in Figure 2.2 give an overall picture of the foot traffic at three different sensors in 2016. Further facetting by day of the week (Figure 2.3) provides a better view of the daily and sub-daily (hourly) pedestrian patterns. Flagstaff Station has a strong commuter pattern, with peaks in the morning and evening, and no pedestrians on the weekend. Around the State Library there are pedestrians walking around during the day, and an unusually large number on one Saturday night and Sunday morning. Birrarung Marr has a varied pedestrian pattern, with very different numbers of people on different days and times.

Faceting, initially called trellis displays (Becker, Cleveland, and Shyu, 1996), is an example of a small multiple (Tufte, 1983), where different subsets of the same data are displayed across one or more conditioning variables. It allows the comparison of subsets. Faceting can also be thought of as a simple ensemble graphic (Unwin and Valero-Mora, 2018). It

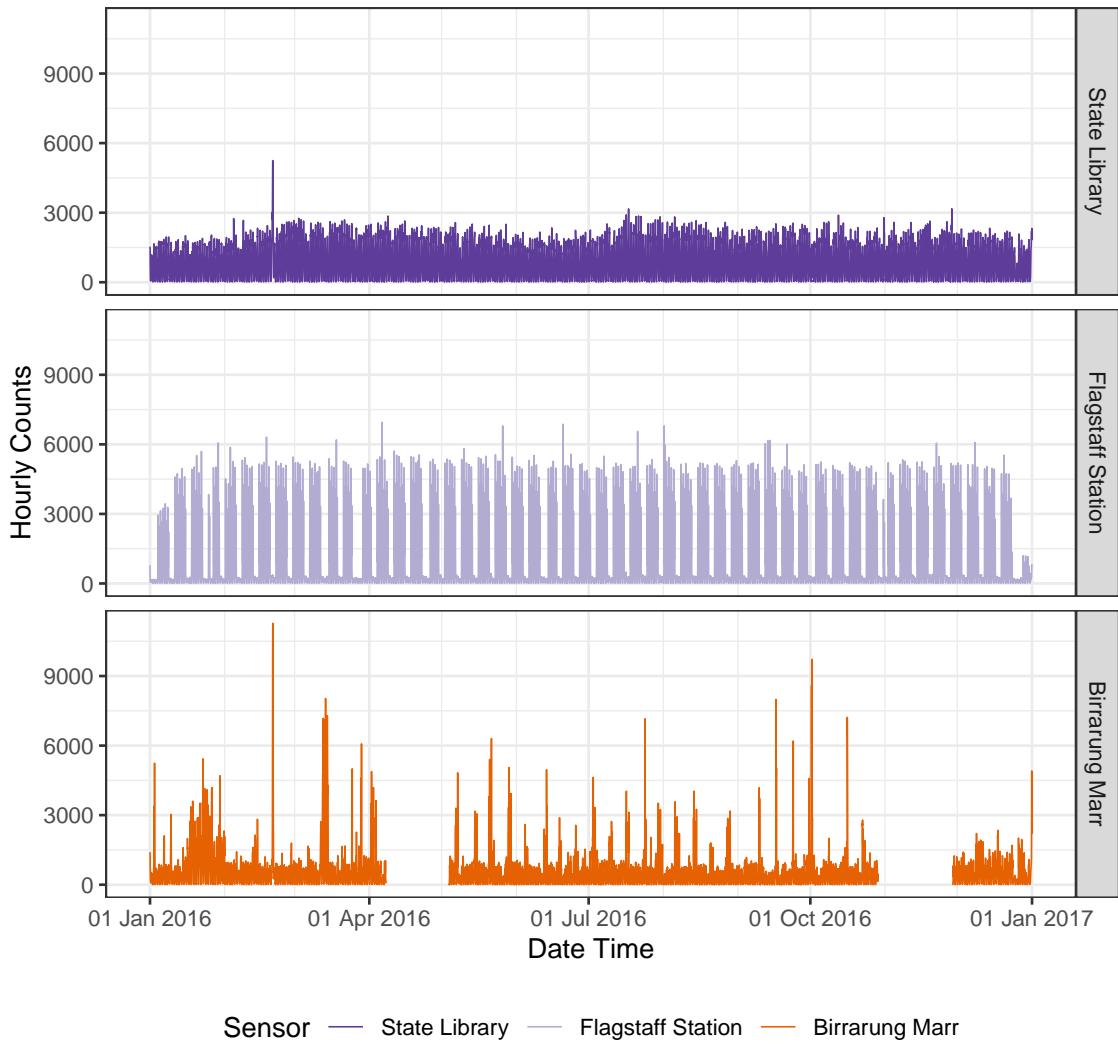


**Figure 2.1:** Google map of the Melbourne city area, grey dots indicate all the sensor locations. The three locations highlighted will be analyzed in the paper: the State Library is a public library; Flagstaff Station is a train station, closed on non-work days; Birrarung Marr is an outdoor park hosting many cultural and sports events.

is a homogeneous collection of plots, whereas the ensemble graphics broadly organize related plots for a data set together into one display.

The work is inspired by Wickham et al. (2012), which uses modular arithmetic to display spatio-temporal data as glyphs on maps. It is also related to recent work by Hafen (2019) which provides methods in the **geofacet** R package to arrange data plots into a grid, while preserving the geographical position. Both of these show data in a spatial context.

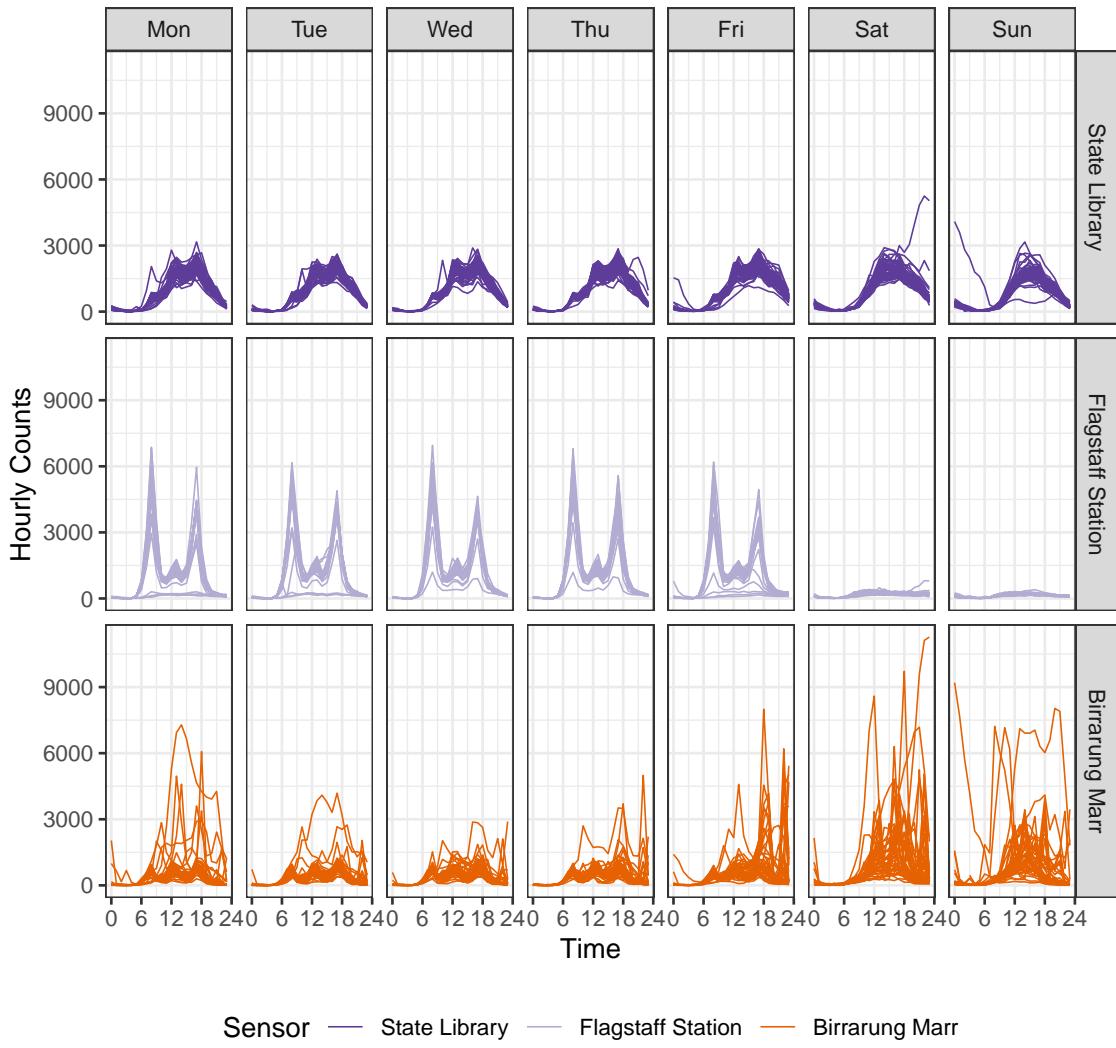
In contrast, calendar-based graphics unpack the temporal variable, at different resolutions, to digest multiple seasonalities and special events. There are some existing works in this area. For example, Van Wijk and Van Selow (1999) developed a calendar view of the heatmap to represent the number of employees in the work place over a year, where colors 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. Jones (2016), Wong (2013), Kothari and Ather (2016), and Jacobs (2017) implemented some variants of calendar-based heatmaps in R packages: **TimeProjection**, **ggTimeSeries**, and **ggtcal** respectively. However, these techniques are limited to color-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 the pedestrian sensor data, will be neglected through daily aggregation. Color-encoding is also low on the hierarchy of



**Figure 2.2:** Time series plots showing 2016 pedestrian counts, measured by three different sensors in the city of Melbourne. Small multiples of lines show that the foot traffic varies at one location to another. The spike in counts at the State Library corresponds to the timing of the event “White Night”, where there were many people taking part in activities in the city throughout the night. A relatively persistent pattern repeats from one week to another at Flagstaff Station. Birrarung Marr looks rather noisy and spiky, with several runs of missing records.

optimal variable mapping (Cleveland and McGill, 1984; Lam, Munzner, and Kincaid, 2007).

The proposed algorithm goes 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



**Figure 2.3:** Hourly pedestrian counts for 2016, faceted by sensors, and days of the week. The focus is on time of day and day of week across the sensors. Daily commuter patterns at Flagstaff Station, the variability of the foot traffic at Birrarung Marr, and the consistent pedestrian behavior at the State Library, can be seen.

enable patterns related to special events more easily pop-up to viewers. The proposed algorithm has been implemented in the `frame_calendar()` and `facet_calendar()` functions in the `sugrrants` package using R.

The remainder of the paper is organized as follows. Section 2.2 details the construction of the calendar layout in depth. It describes the algorithms of data transformation (Section 2.2.1), the available options (Section 2.2.2), and variations of its usage (Section 2.2.3). Section 2.2.3 explains the full faceting extension, that is equipped with formal labels and axes. An analysis of half-hourly household energy consumption, using the calendar

display, is illustrated in a case study in Section 2.3. Section 2.4 discusses the limitations of calendar displays and possible new directions.

## 2.2 Creating a calendar display

### 2.2.1 Data transformation

The algorithm of transforming data 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 conforms to a layout of 5 rows and 7 columns, 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 deterministic given the year of the calendar. Months are of different lengths, ranging from 28 to 31 days. Some months could extend over six weeks, but for these months the last few days are wrapped up to the top row of the block for compactness, and because it is convention. The notation for creating these cells is as follows:

- $k = 1, \dots, 7$  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.
- $g = k, \dots, (k + d)$  indexes the day in the month, inside the 35 possible cells.

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

$$\begin{aligned} i &= \lceil (g \bmod 35) / 7 \rceil, \\ j &= g \bmod 7. \end{aligned} \tag{2.1}$$

Figure 2.4 illustrates this  $(i, j)$  layout for a month where  $k = 5$ .

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

**Figure 2.4:** Illustration of the indexing layout for cells in a month, where  $k$  is day of the week,  $g$  is day of the month,  $(i, j)$  indicates grid position.

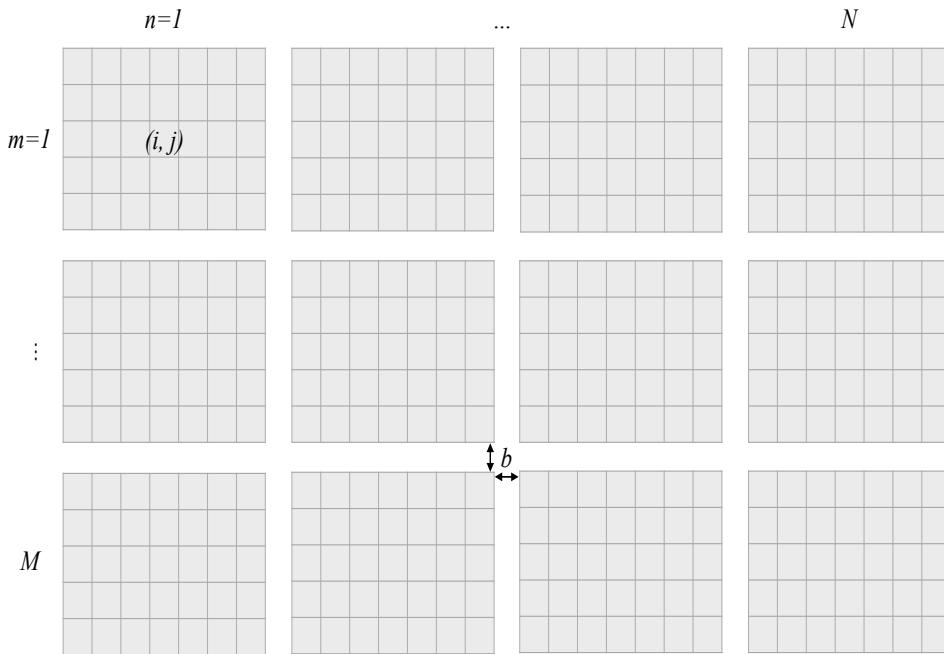
To create the layout for a full year,  $(m, n)$  denotes the position of the month arranged in the plot, where  $1 \leq m \leq M$  and  $1 \leq n \leq N$ ;  $b$  denotes the small amount of white space between each month for visual separation. Figure 2.5 illustrates this layout where  $M = 3$  and  $N = 4$ .

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  $c$  be the scaled count.

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

$$\begin{aligned} x &= j + (n - 1) \times 7 + (n - 1) \times b + h, \\ y &= i - (m - 1) \times 5 - (m - 1) \times b + c. \end{aligned} \tag{2.2}$$

Note that for the vertical direction, the top left is the starting point of the grid (in Figure 2.4), which is easier to lay out and why the subtraction is performed. Within each cell, the starting position is the bottom left.

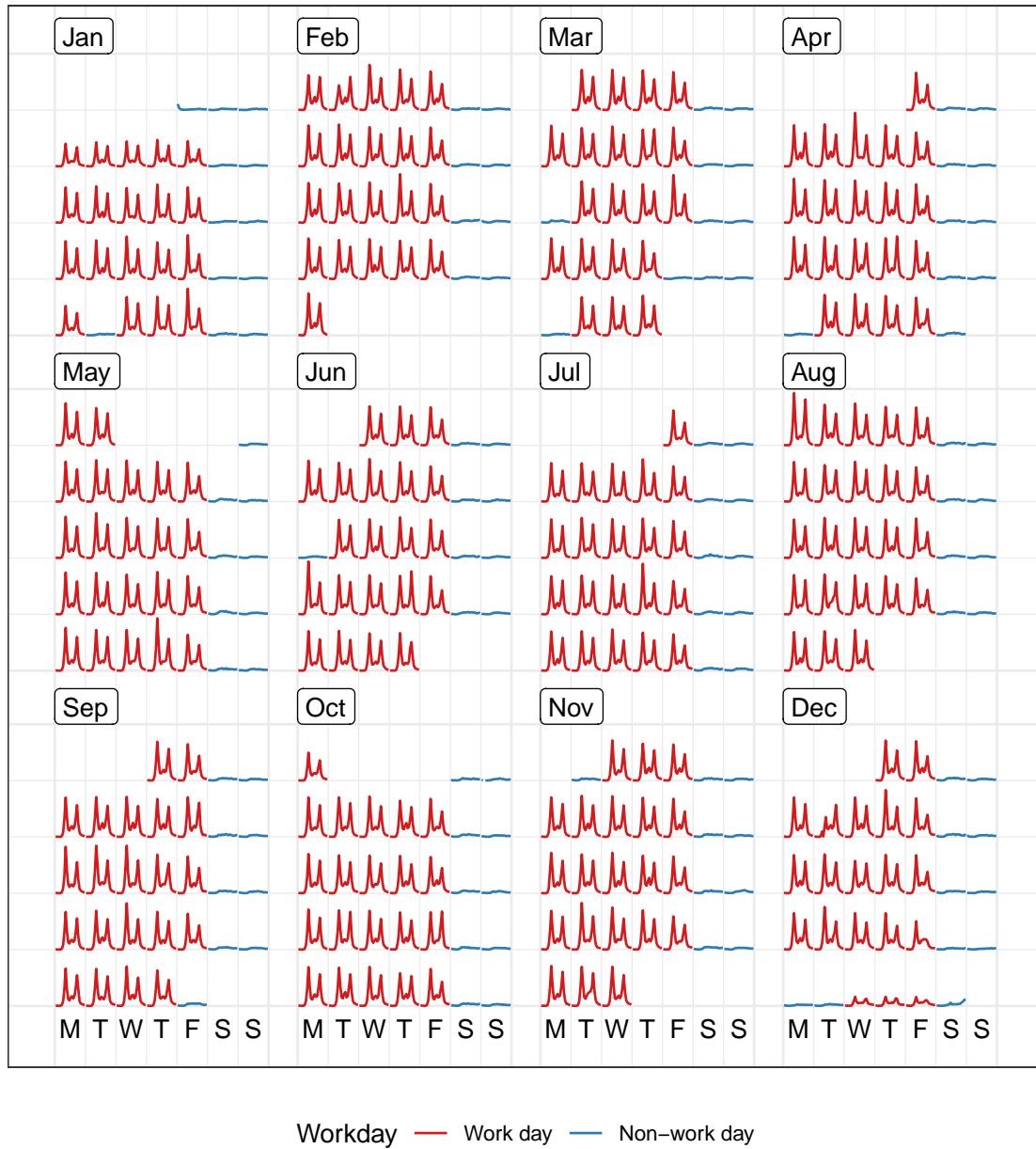


**Figure 2.5:** Illustration of the indexing layout for months of one year, where  $M$  and  $N$  indicate number of rows and columns,  $b$  is a space parameter separating cells.

Figure 2.6 shows the line glyphs framed in the monthly calendar over the year 2016. This is achieved by the `frame_calendar()` function, which computes the coordinates on the calendar for the input data variables. These can then be plotted using the usual `ggplot2` R package (Wickham et al., 2019a). Thus, the grammar of graphics can be applied.

In order to make calendar-based graphics more accessible and informative, reference lines dividing each cell and block, as well as labels indicating week day and month are also computed before plot 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 2.6, 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  $\min(y)$ .



**Figure 2.6:** The calendar plot of hourly foot traffic at Flagstaff Station, using line glyphs. The disparities between week day and weekend along with public holiday, are immediately apparent. The arrangement of the data into a  $3 \times 4$  monthly grid represents all the traffic in 2016. Note that, the algorithm wraps the last few days in the sixth week to the top row of each month block for a compact layout, which occurs to May and October.

The month labels located on the top left using  $(\min(x), \max(y))$  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  $j$ .

## 2.2.2 Options

The algorithm has several optional parameters that modify the layout, direction of display, scales, plot size and switching to polar coordinates. These are accessible to the user by the inputs to the function `frame_calendar()`:

```
frame_calendar(data, x, y, date, calendar = "monthly", dir = "h",
  week_start = 1, nrow = NULL, ncol = NULL, polar = FALSE, scale = "fixed",
  width = 0.95, height = 0.95, margin = NULL)
```

It is assumed that the `data` is in tidy format (Wickham, 2014), and `x`, `y` are the variables that will be mapped to the horizontal and vertical axes in each cell. For example, the `x` is the time of the day, and `y` is the count (Figure 2.6). The `date` argument specifies the date variable used to construct the calendar layout.

The algorithm handles displaying a single month or several years. The arguments `nrow` and `ncol` specify the layout of multiple months. For some time frames, some arrangements may be more beneficial than others. For example, to display data for three years, setting `nrow = 3` and `ncol = 12` would show each year on a single row.

## Layouts

The monthly calendar is the default, but two other formats, weekly and daily, are available with the `calendar` argument. The daily calendar arranges days along a row, one row per month. The weekly calendar stacks weeks of the year vertically, one row for each week, and one column for each day. The reader can scan down all the Mondays of the year, for example. The daily layout puts more emphasis on day of the month. The weekly calendar is appropriate if most of the variation can be characterized 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. Temporal patterns motivate which variant should be employed.

## Orientation

By default, grids are laid out horizontally. This can be transposed by setting the `dir` parameter to "v", in which case  $i$  and  $j$  are swapped in Equation (2.1). This can be useful for creating calendar layouts for countries where vertical layout is the convention.

## Start of the week

The start of the week for a monthly calendar is adjustable. The default is Monday (1), which is chosen from the data perspective. The week, however, can begin with Sunday (7) as commonly used in the US and Canada, or other week day, subject to different countries and cultures.

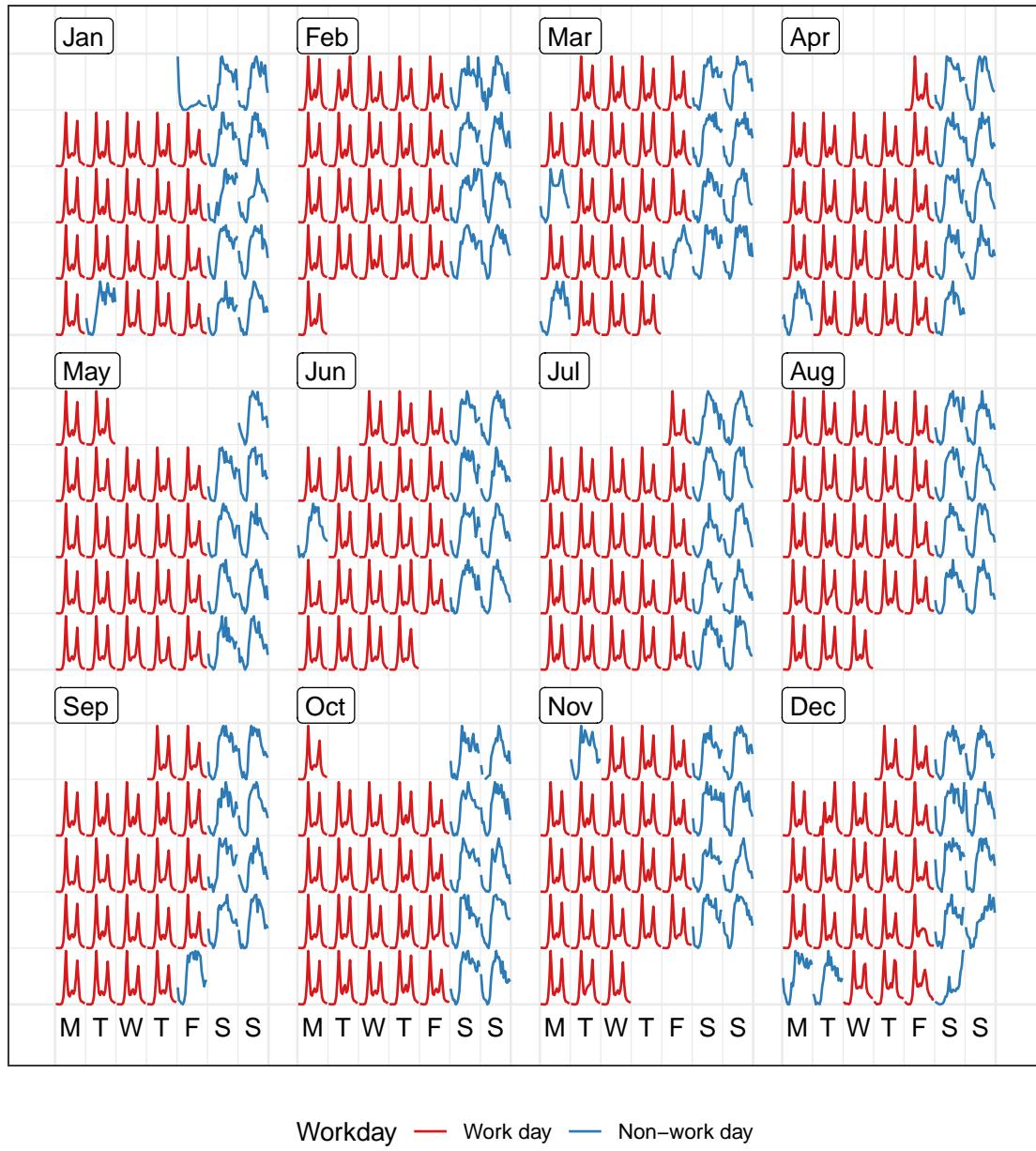
## Polar transformation

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). This produces star glyphs (Chambers et al., 2017), where time series lines are transformed in polar coordinates, embedded in the monthly calendar layout.

## Scales

By default, global scaling is done for values in each plot, with the global minimum and maximum used to fit values into each cell. If the emphasis is on comparing trend rather than magnitude, it is useful to scale locally. For temporal data, this would harness the temporal components. The choices 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.

With local scaling, the overall variation gives way to the individual shape. Figure 2.7 shows the same data as Figure 2.6, scaled locally using `scale = "free"`. The daily trends are magnified.



**Figure 2.7:** Line glyphs on the calendar format showing hourly foot traffic at Flagstaff Station, scaled individually by day. The shape on a single day becomes more distinctive, as compared to Figure 2.6.

The `free_wday` scales each week day together. It can be useful to compare trends across week days, allowing relative patterns for weekends versus week days to be examined. Similarly, the `free_mday` uses free scaling for any day within a given month.

### Language support

Most countries have adopted this western calendar layout, while the languages used for week day and month would be different across countries. Other language specifications than English, for text labeling, are available.

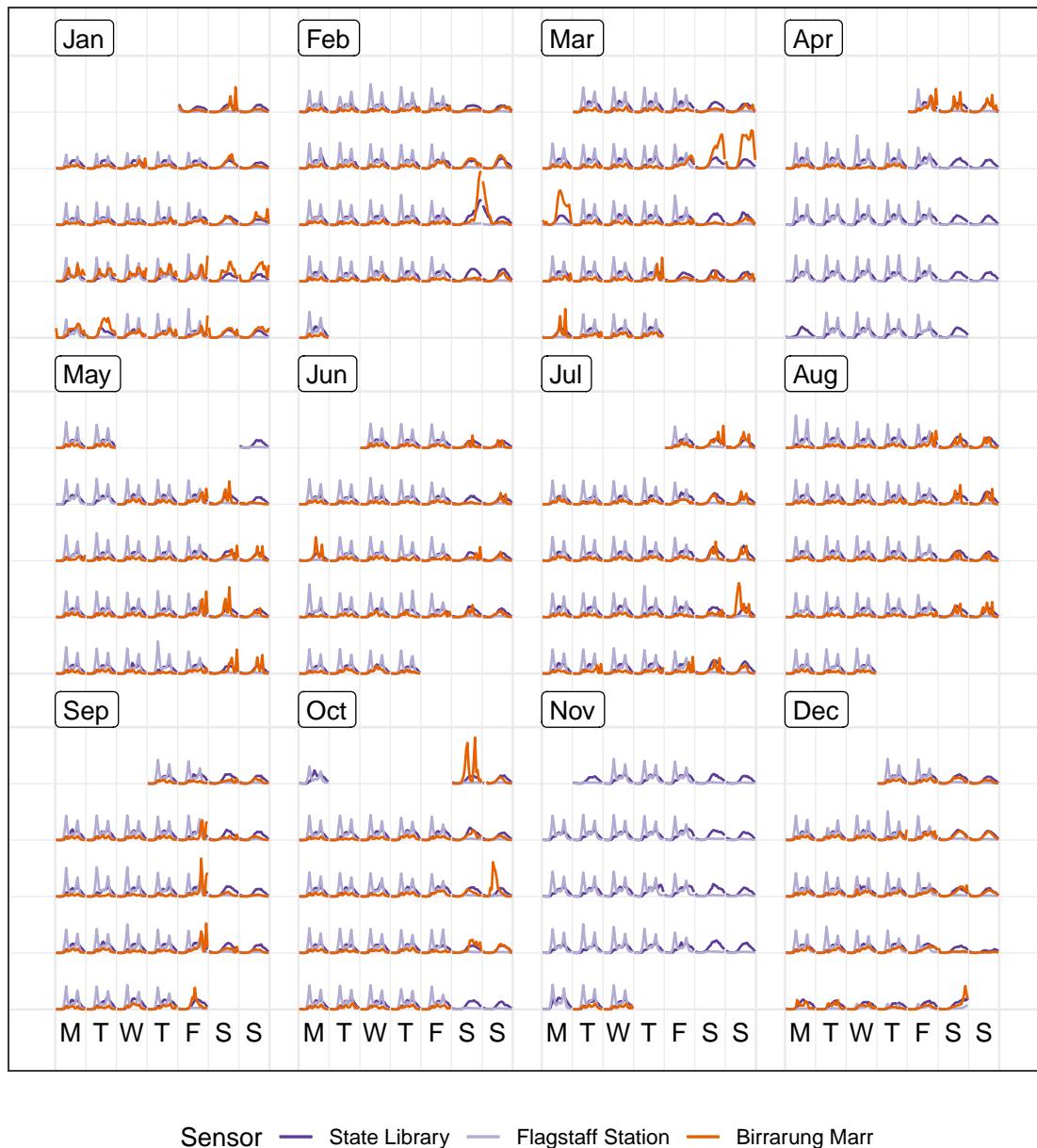
### 2.2.3 Varieties of calendar display

#### Information overlay

Plots can be layered. A comparison of sensors can be done by overlaying them in the same calendar pane. Figure 2.8 overlays the pedestrian counts for three locations on the same calendar. Differences between the pedestrian patterns at these locations can be more directly compared. For example, the magnitude of the difference in pedestrians at Flagstaff Station at peak hours of commuter can be seen. The big peak in pedestrian counts for special events at Birrarung Marr is clear. Birrarung Marr has a very distinct temporal pattern relative to the other two locations. The nighttime events, such as White Night (third Saturday in February), only affects the foot traffic at the State Library and Birrarung Marr.

#### Faceting by covariates

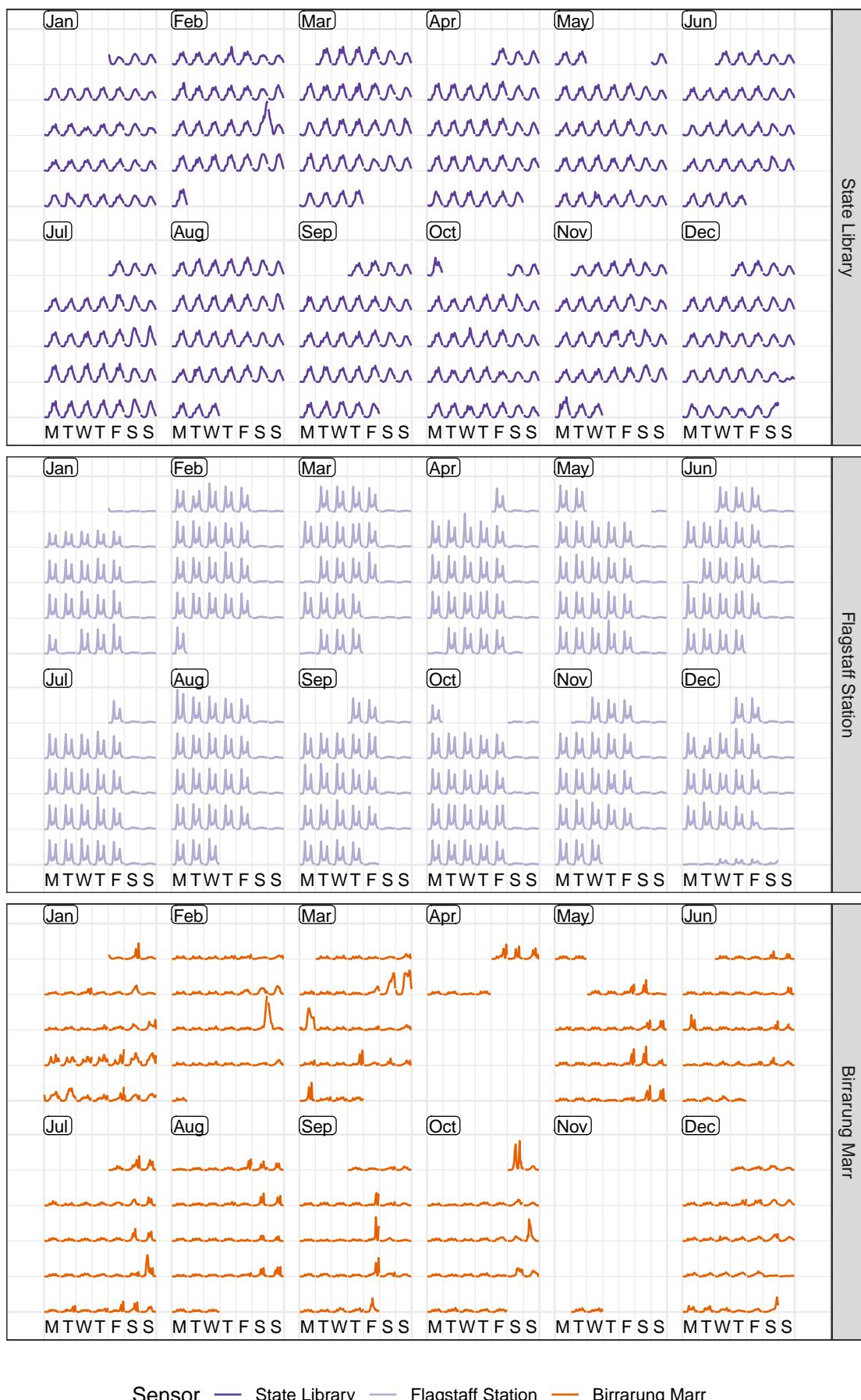
To avoid overlapping, when differences between groups are large enough to be seen separately, the calendar layout can be faceted into a series of subplots for the different sensors. Figure 2.9 shows calendar plots that are faceted by sensors. This arrangement allows comparison of the overall structure between sensors, while emphasizing individual sensor variation. In particular, it can be immediately learned that Birrarung Marr was busy and packed over many weekends, but events took place on Friday evenings only in September. The Australian Open, a major international tennis tournament, attracted constant foot traffic in the last two weeks of January. The calendar plot can be faceted by any categorical variable in the data.



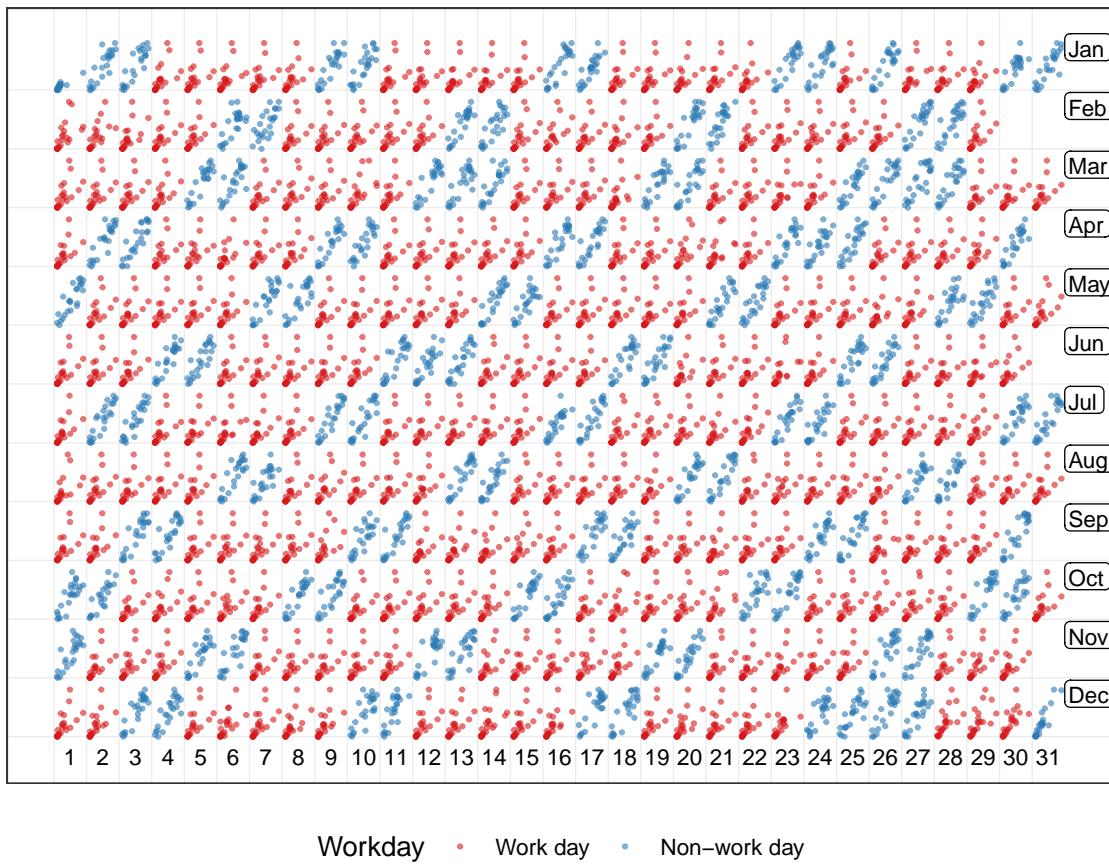
**Figure 2.8:** Overlaying line graphs of the three sensors in the monthly calendar, to enable a direct comparison of the counts at three locations. They have very different traffic patterns. Birrarung Marr tends to attract large numbers of pedestrians for special events typically held on weekends, contrasting to the bimodal massive peaks showing commuting traffic at Flagstaff Station.

### Different types of plots

Almost any type of plot can be shown in a calendar pane. Most of the full range of plotting capabilities in `ggplot2` is available. An example is shown in Figure 2.10: the panes contain lag scatterplots, constructed with local scaling for each day at Flagstaff



**Figure 2.9:** Line charts, embedded in the  $6 \times 2$  monthly calendar, colored and faceted by the sensors. The variations of an individual sensor are emphasized.



**Figure 2.10:** Lag scatterplots, with local scaling, in the daily calendar layout. Each hour's count is plotted against the previous hour's count at Flagstaff Station, to demonstrate the autocorrelation at lag 1. The correlation between them is more consistent on non-work days than work days.

Station, where the lagged hourly count is assigned to the `x` argument and the current hourly count to the `y` argument. It indicates strong autocorrelation on weekends, and weak autocorrelation on work days. The V-shape in the week day graphs arises when the next hour sees a substantial increase or decrease in counts. This is due to the morning and afternoon commuter patterns in Figure 2.6: the adjacent hours are positively correlated when approaching to the peak hour but negatively correlated when moving away from the peak.

The algorithm can also produce more complicated plots, such as boxplots. Figure 2.11 uses a loess smooth line (Cleveland, 1979) 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

(December 24), go shopping on the Boxing day (December 26), and stay out for the fireworks on New Year's Eve. The text in the plot is labeled in Chinese, showcasing the support for other languages.

### Interactivity

As a data restructuring tool, the interactivity of calendar-based displays can be easily enabled, as long as the interactive graphics system remains true to the spirit of the grammar of graphics, for example, `plotly` (Sievert, 2018) in R. As a standalone display, an interactive tooltip can be added to show labels when mousing over a point in the calendar plot, for example the hourly count with the time of day. It is difficult to sense the values from the static display, but the tooltip makes it possible. Options in the `frame_calendar()` function can be ported to a form of selection button or text input in a graphical user interface like R shiny (Chang et al., 2019). The display will update on the fly accordingly, via clicking or text input, as desired.

Linking calendar displays to other types of charts is valuable to visually explore the relationships between variables. An example can be found in the `wanderer4melb` shiny application (Wang, 2019). The calendar most naturally serves as a tool for date selection: by selecting and brushing the glyphs in the calendar, it subsequently highlights the elements of corresponding dates in other time-based plots. Conversely, selecting on weather data plots, linked to the calendar can help to assess if very hot/cold days and heavy rain, affect the number of people walking in downtown Melbourne. The linking between weather data and calendar displays is achieved using the common dates.

### Faceted calendar

The `frame_calendar()` function described in Section 2.2.2 is a data restructuring function, neatly integrating into a data pipeline but it requires two steps: data transformation and then plot. There is also little freedom to tailor axes and labels, because specialist code needs to be applied.

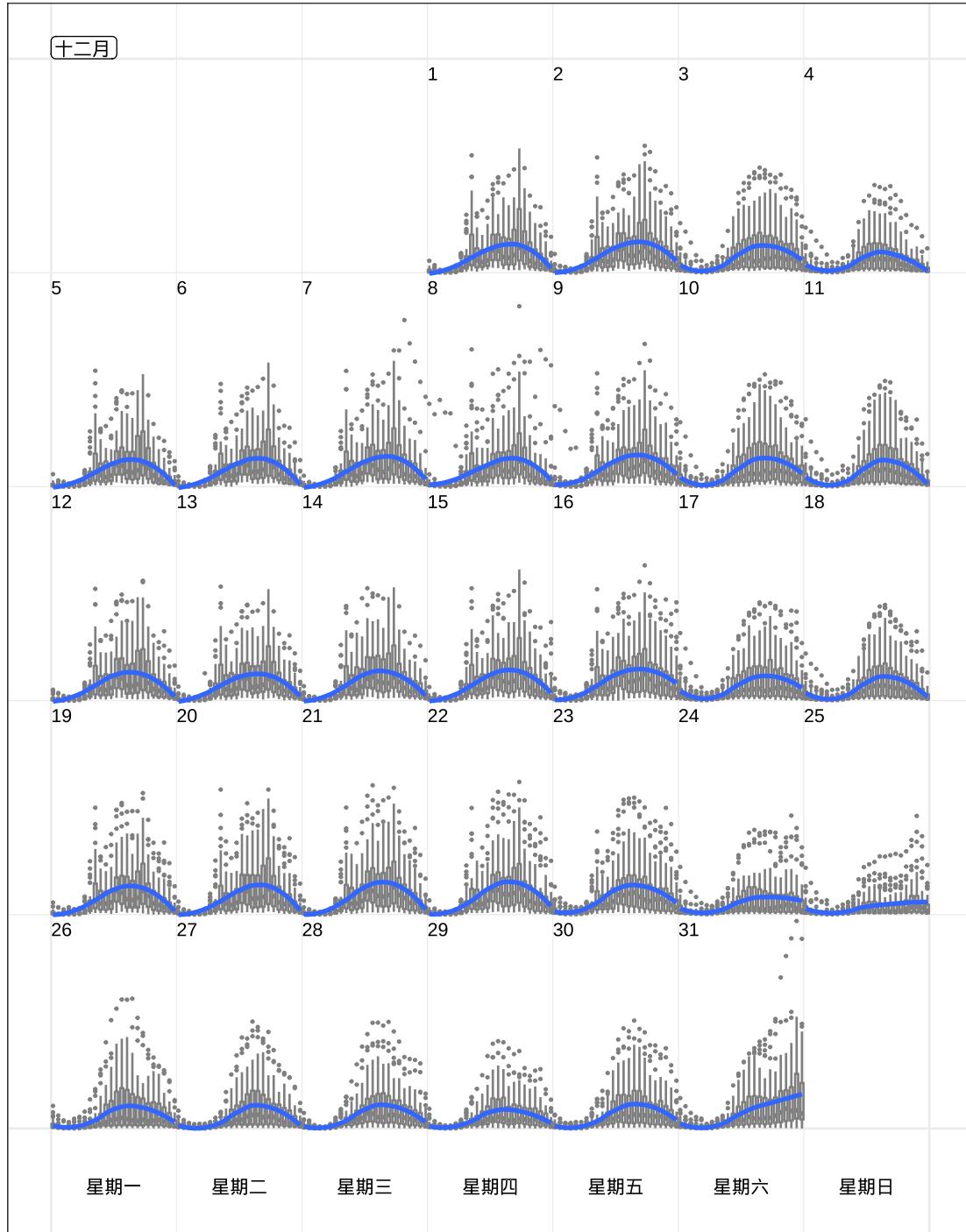
The `facet_calendar()` integrates the algorithm into the `ggplot2` graphical system so that the calendar layout is automatic, and the full functionality of axes, labels, and customization is accessible. A faceting method lays out panels in a grid. The user needs to supply the variable containing dates, in order for the facetting calendar function to prepare the arrangement of panels, as defined by Equation (2.1). The remainder of the plot construction for each panel is handled entirely by `ggplot2` internals.

Formal axes and labels unavailable in calendar plots generated by the `frame_calendar()` are possible (Figure 2.12). It is much easier for readers to infer the scaling (global or local) employed for the plot. Non-existing panels mean non-existing days in the month, and blank panels indicate missing data on the day. This avoids confusion about missing data or days when missingness lives in the ends of month panels, which may occur when using `frame_calendar()`.

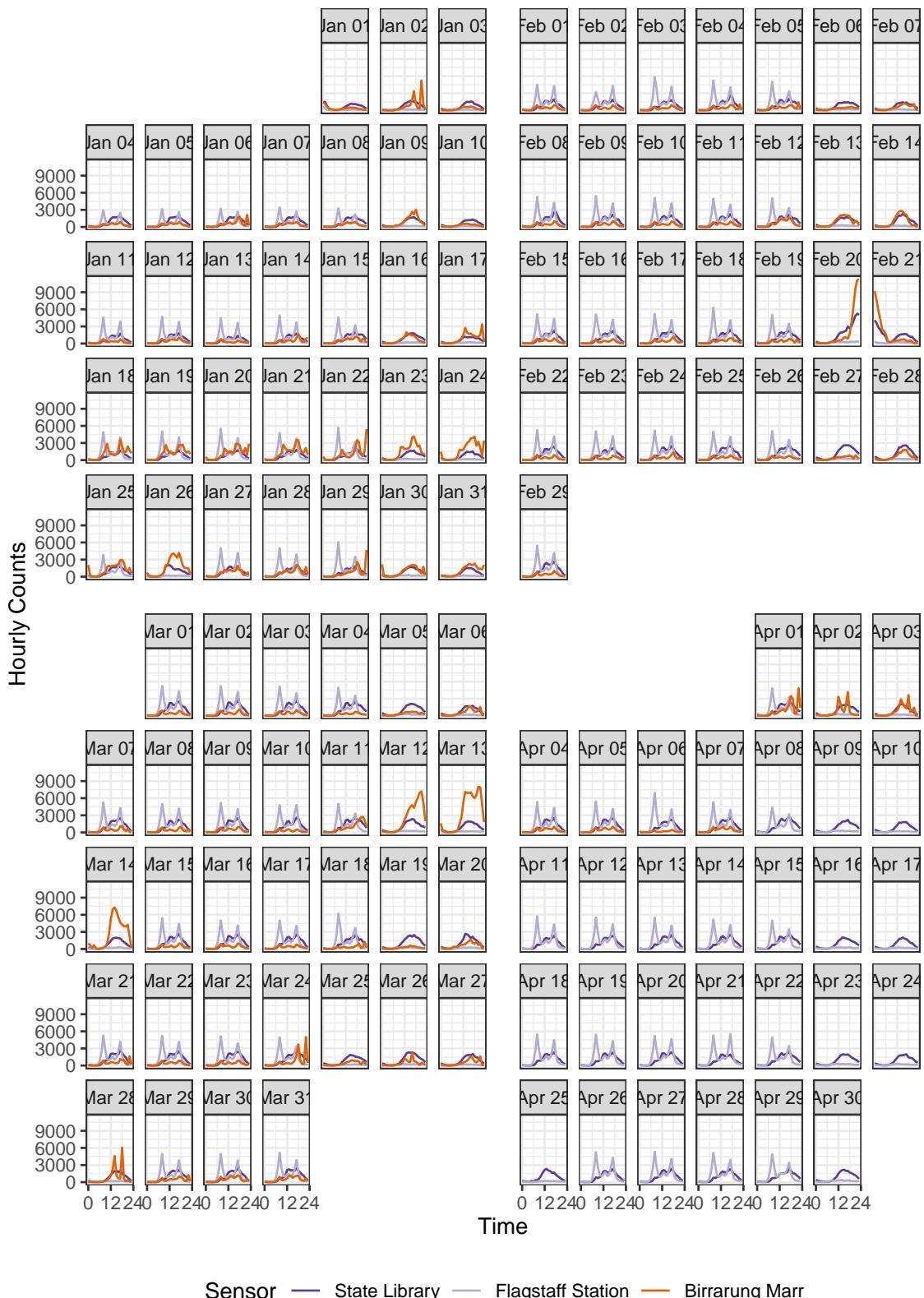
However, the `facet_calendar()` takes much more run time compared with `frame_calendar()`. The faceted calendar also uses more plot real estate for panel headings and axes. The reader can compare the two approaches by examining the compact Figure 2.8, relative to Figure 2.12. The space consumed by the former shows a full year, and the latter shows four months, only a third of the data. For fast rendering and economy of space, `frame_calendar()` is recommended.

#### 2.2.4 Reasons to use calendar-based graphics

The purpose of the calendar display is to facilitate quick discoveries of unusual patterns in people's activities, which is consistent with why analysts should and do use data visualization. It complements the traditional graphical toolbox used to understand general trends, and better profiles vivid and detailed data stories about the way we live. Comparing the conventional displays (Figure 2.2 and 2.3) with the new display (Figure 2.9), it can be seen that the calendar display is more informatively compelling: when special events happened, and on what day of the week, and whether they were day or night events. For example, Figure 2.9 informs the reader that many events were held in Birrarung Marr on weekend days, while September's events took place on Friday evenings, which is difficult to discern from conventional displays.



**Figure 2.11:** Side-by-side boxplots of hourly counts for all the 43 sensors in December 2016, with the loess smooth line superimposed on each day. It shows the hourly distribution in the city as a whole. The increased variability is notable on the last day of December as New Year's Eve approaches. The month and week day are labeled in Chinese, which demonstrates the support for languages other than English.



**Figure 2.12:** A faceted calendar showing a fraction of the data shown in Figure 2.8. The faceted calendar takes more plot real estate than the calendar plot, but it provides native ggplot2 support for labels and axes.

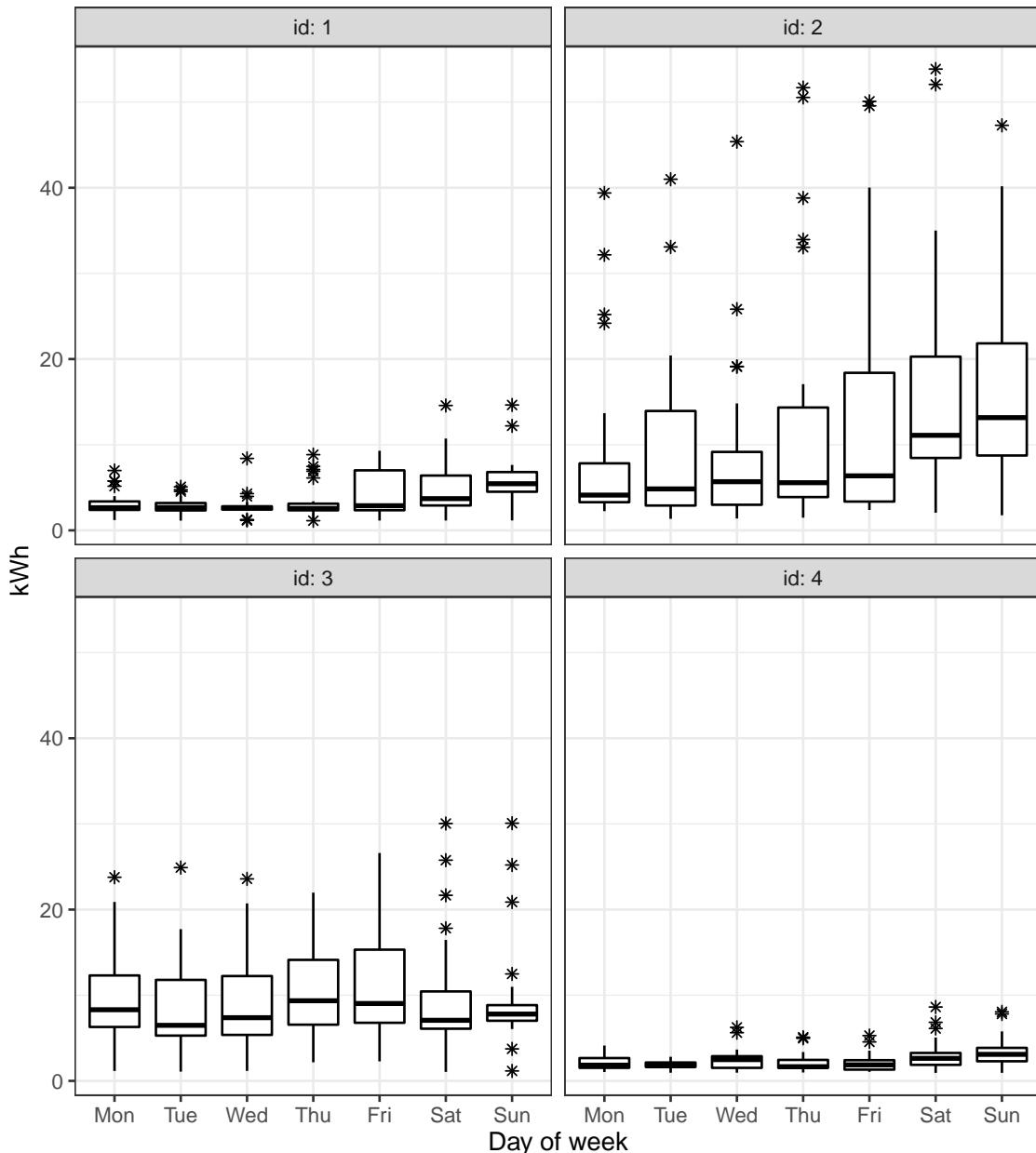
## 2.3 Case study

The use of the calendar display is illustrated on smart meter energy usage from four households in Melbourne, Australia. Individuals can download their own data from the energy supplier, and the data used in this section is sourced from four colleagues of the authors. The calendar display is useful to help people understand their energy use. The data contains half-hourly electricity consumption in the first half of 2018. The analysis begins by looking at the distribution over days of week, then time of day split by work days and non-work days, followed by the calendar display to inspect the daily schedules.

Figure 2.13 shows the energy use across days of week using boxplots. Inspecting the medians across households tells us that household 3, a family size of one couple and two children, uses more energy over the week days than other households. The relatively larger boxes for household 2 indicate greater variability in daily energy consumption with noticeable variations on Thursdays, and much higher usage over the weekends. The other two households (1 and 4) tend to consume more energy with more variation on the weekends relative to the week days, reflecting work and leisure patterns.

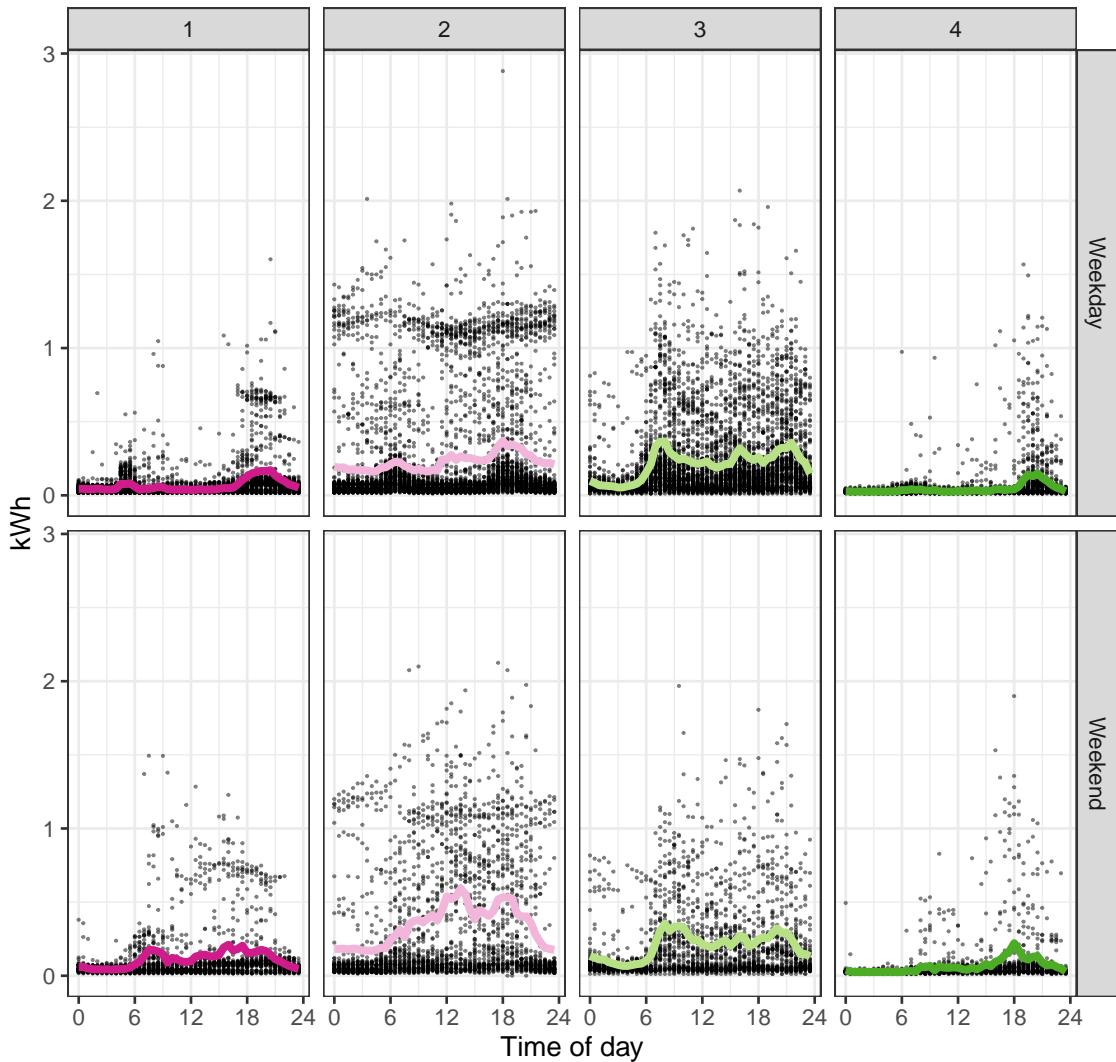
Figure 2.14 shows energy consumption against time of day, separately by week day and weekend. Household 1 is an early riser, starting their day before 6am and going back home around 6pm on week days. They switch air conditioning on when they get home from work and keep it operating until midnight, evident from the small horizontal cluster of points around 0.8 kWh. On the other hand, the stripes above 1 kWh for household 2 indicates that air conditioning may run continuously for some periods, consuming twice the energy as household 1. A third peak occurs around 3pm for household 3 only, likely coinciding when the children arrive home from school. They also have a consistent energy pattern between week days and weekends. As for household 4, their home routine starts after 6pm on week days. Figures 2.13 and 2.14, part of a traditional graphical toolkit, are useful for summarizing overall deviations across days and households.

Figure 2.15 displays the global scaling of each household's data in a calendar layout, unfolding their day-to-day life via electricity usage. Glancing over household 1, their overall energy use is relatively low. Their week day energy use is distinguishable from



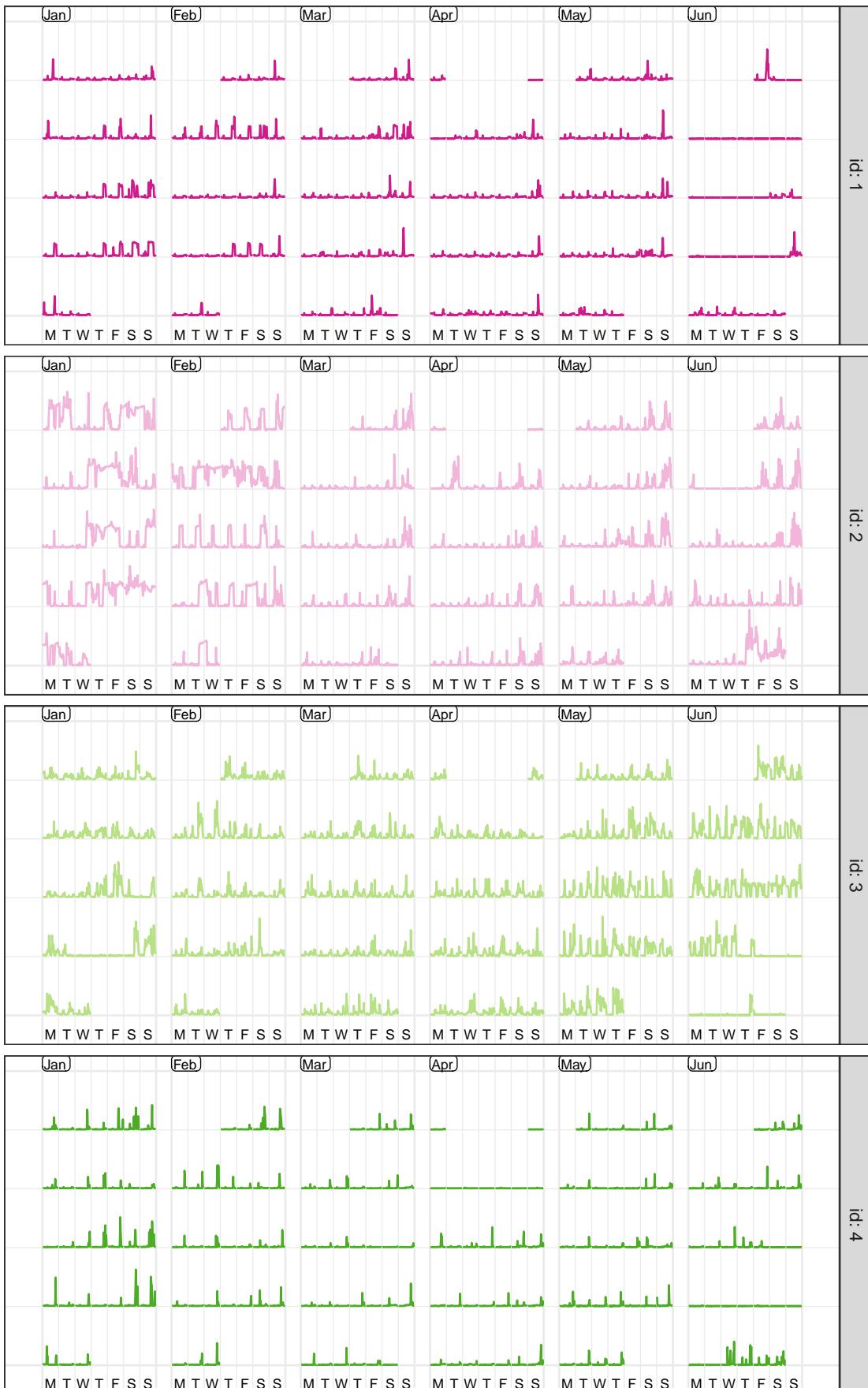
**Figure 2.13:** Boxplots of daily energy usage against day of week for four households. Suggested by the medians, household 3 uses more energy than the others on the week days. By contrast, household 2 sees considerably larger variability.

their weekends, indicating a working household. The air conditioner appears to be used in the summer months (January and February) for a couple of hours in the evening and weekends. In contrast, household 2 keeps a cooling system functioning for much longer hours, which becomes more evident from late Wednesday through Thursday to early Friday in mid-January. These observations help to explain the stripes and clusters of household 2 in Figure 2.14. It is difficult to give a succinct description of household 3



**Figure 2.14:** Scatterplots of half-hourly energy usage by time of day, with hourly averages overlaid, faceted by household and type of day. All households have different week days versus weekends daily routines. On week days, household 1 wakes up early before 6am, and household 2 around 6am, followed by household 3 and 4. The use of air conditioning is notable in households 1 and 2, as seen by horizontal clusters of points.

since everyday energy pattern is variable, but May and June see more structure than the previous months. Individual data can be idiosyncratic, hence aggregated plots like Figure 2.13 and 2.14 are essential for assembling pieces to form a picture. However, the calendar plots tell the stories that are untold by previous plots, for example, their vacation time. Household 1 is on vacation over three weeks of mid-June, and household 2 also took some days off in the second week of June. Further, household 3 takes one short trip in January and another one starting in the fourth week of June. Household 4 is away over two or



— 1 — 2 — 3 — 4

**Figure 2.15:** Calendar displays faceted by each household using global scales. Long flat low energy usage indicates vacation time, and high energy consumption by household 2 is visible in January and February.

three weeks in early April and late June. They all tend to take breaks during June probably due to the fact that the University winter break starts in June.

## 2.4 Discussion

The calendar-based visualization provides data plots in the familiar format of an everyday tool. Patterns relating to special events and public holidays for the region are more visible to the viewer.

The calendar layout will be useful for studying consumer trends and human behavior. It will be less useful for physical processes such as weather. The layout does not replace traditional displays, but serves to complement them 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 methodology creates the western calendar layout, because most countries have adopted this format. The main difference between countries is the use of different languages for labeling, which is supported by the software. Formats beyond the western calendar, or six-weeks and tetris-like layouts could be achieved by slightly tweaking the modular arithmetic approach. These features will be added as new options in the future.

## Acknowledgements

We would like to thank Stuart Lee and Heike Hofmann for their feedback on earlier versions of this work. We thank Thomas Lin Pedersen for pointing out some critical **ggplot2** internals, which makes the `facet_calendar()` implementation possible. We are very grateful to anonymous reviewers for helpful comments that have led to many improvements of the paper. The **sugrrants** R package is available from CRAN <https://CRAN.R-project.org/package=sugrrants> and the development version is available on Github <https://github.com/earowang/sugrrants>. All materials required to reproduce this article and a history of the changes can be found at the project's Github repository <https://github.com/earowang/paper-calendar-vis>.

## **Chapter 3**

# **A new tidy data structure to support exploration and modeling of temporal data**

Mining temporal data for information is often inhibited by a multitude of formats: regular or irregular time intervals, point events that need aggregating, multiple observational units or repeated measurements on multiple individuals, and heterogeneous data types. This work presents a cohesive and conceptual framework for organizing and manipulating temporal data, which in turn flows into visualization, modeling and forecasting routines. Tidy data principles are extended to temporal data by: (1) mapping the semantics of a dataset into its physical layout; (2) including an explicitly declared “index” variable representing time; (3) incorporating a “key” comprising single or multiple variables to uniquely identify units over time. This tidy data representation most naturally supports thinking of operations on the data as building blocks, forming part of a “data pipeline” in time-based contexts. A sound data pipeline facilitates a fluent workflow for analyzing temporal data. The infrastructure of tidy temporal data has been implemented in the R package, called **tsibble**.

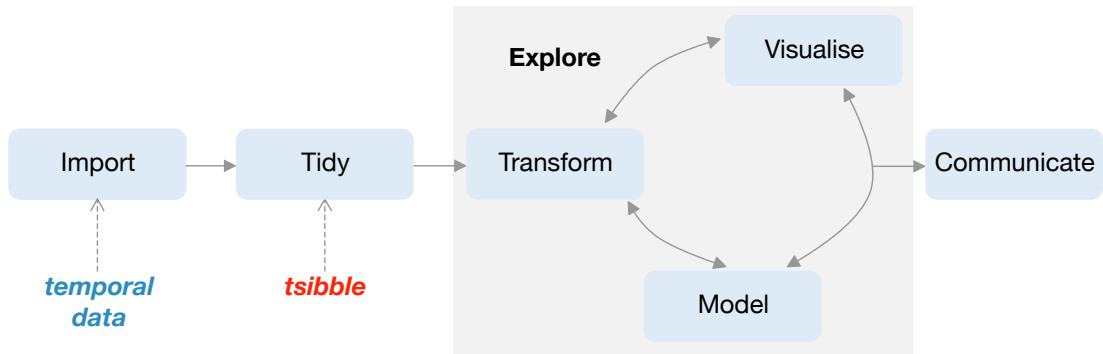
## 3.1 Introduction

Temporal data arrives in many possible formats, with many different time contexts. For example, time can have various resolutions (hours, minutes, and seconds), and can be associated with different time zones with possible adjustments such as daylight saving time. Time can be regular (such as quarterly economic data or daily weather data), or irregular (such as patient visits to a doctor’s office). Temporal data also often contains rich information: multiple observational units of different time lengths, multiple and heterogeneous measured variables, and multiple grouping factors. Temporal data may comprise the occurrence of time-stamped events, such as flight departures.

Perhaps because of this variety and heterogeneity, little organization or conceptual oversight on how one should get the wild data into a tamed state is available for temporal data. Analysts are expected to do their own data preprocessing and take care of anything else needed to allow further analysis, which leads to a myriad of ad hoc solutions and duplicated efforts.

Wickham and Gromelund (2016) proposed the tidy data workflow, to give a conceptual framework for exploring data (as described in Figure 3.1). In the temporal domain, data with time information arrives at the “import” stage. A new abstraction, *tsibble*, introduced in this paper, is the gatekeeper at the “tidy” stage, to verify if the raw temporal data is appropriate for downstream analytics. The exploration loop will be aided with declarative grammars, yielding more robust and accurate analyses.

The paper is structured as follows. Section 3.2 reviews temporal data structures corresponding to time series and longitudinal analysis, and discusses “tidy data”. Section 3.3 proposes contextual semantics for temporal data, built on top of tidy data principles. The concept of data pipelines, with respect to the time domain, is discussed in depth in Section 3.4, followed by a discussion of the design choices made in the software implementation in Section 3.5. Two case studies are presented in Section 3.6 illustrating temporal data exploration using the new infrastructure. Section 3.8 summarises current work and discusses future directions.



**Figure 3.1:** Annotation of the data science workflow regarding temporal data, drawn from Wickham and Grolemund (2016). The new data structure, *tsibble*, makes the connection between temporal data input, and downstream analytics. It provides elements at the “tidy” step, which produce tidy temporal data for temporal visualization and modeling.

## 3.2 Data structures

### 3.2.1 Comparing time series and longitudinal data

Temporal data problems often fall into two types of analysis, time series and longitudinal. Both of these may have similar data input, but the representation for modelling is typically different. Time series analysis tends to focus on the dependency within series, and the cross-correlation between series. Longitudinal analysis tends to focus on overall temporal patterns across demographic or experimental treatment strata, that incorporates within subject dependency.

Time series can be univariate or multivariate, and require relatively long lengths (i.e., large  $T$ ) for modeling. With this large  $T$  property, the series can be handled as stochastic processes for the primary purposes of forecasting and characterizing temporal dynamics. Due to an expectation of regularly spaced time, and equal lengths across series, multivariate time series are typically assumed to be in the format where each column contains a single time series, and time is specified implicitly. This also implies that data are columns of homogeneous types: either all numeric or all non-numeric. It can be frustrating to wrestle data from its original format to this modeling format. The format could be considered to be model-centric, rather than data-centric, and thus throws the analyst into the deep end of the pool, rather than allowing them to gently wade to the modeling stage from the

shallow end. The expectation is that the “model” is at the center of the analytical universe. This is contrary to the **tidyverse** conceptualization (Figure 3.1), which holistically captures the data workflow. More support needs to be provided, in the form of consistent tools and data structures, to transform the data into the analytical cycle.

Longitudinal data (or panel data) typically assumes fewer measurements (small  $T$ ) over a large number of individuals (large  $N$ ). It often occurs that measurements for individuals are taken at different time points, and in different quantities. The primary format required for modeling is stacked data, blocks of measurements for each individual, with columns indicating panels, times of measurement and the measurements themselves. An appealing feature is that data is structured in a semantic manner with reference to observations and variables, with panel and time explicitly stated.

### 3.2.2 Existing data standards

In R (R Core Team, 2018), time series and longitudinal data are of different representations. The native **ts** object and the enhancements of **zoo** (Zeileis and Grothendieck, 2005) and **xts** (Ryan and Ulrich, 2018), assemble time series into wide matrices with implicit time indexes. If there are multiple sub-groups, such as country or product type, these would be kept in different data objects. A relatively new R package **tibbletime** (Vaughan and Dancho, 2018b) proposed a data class of *time tibble* to represent time series in heterogeneous long format. It only requires an index variable to be declared. However, this is insufficient, and a more rigid data structure is required for temporal analytics and modeling. The **plm** (Croissant and Millo, 2008) and **panelr** (Long, 2019) packages both manage longitudinal data in long format.

Stata (StataCorp, 2017) provides two commands, **tsset** and **xtset**, to declare time series and panels respectively, both of which require explicit panel id and time index specification. Different variables would be stored in multiple columns. The underlying data arrangement is only long form, for both types of data. Both groups of functions can be applied interchangeably to whether the data is declared for time series or longitudinal data. The SAS software (SAS Institute Inc., 2018) also handles both types of data in the same way as Stata.

### 3.2.3 Tidy data

Wickham (2014) coined the term “tidy data”, to standardize the mapping of the semantics of a dataset to its physical representation. In tidy form, rows correspond to observations and columns to variables. Tidy data is a rephrasing of the second and third normal forms from relational databases, but the explanation in terms of observations and variables is easier to understand because it uses statistical terminology.

Multiple time series, with each column corresponding to a measurement is tidy data when the time index is explicitly stored in a column. The stacked data format used in longitudinal data is tidy, and accommodates explicit identification of sub-groups.

The tidy data structure is the fundamental unit of the **tidyverse**, which is a collection of R packages designed for data science. The ubiquitous use of the **tidyverse** is testament to the simplicity, practicality and general applicability of the tools. The **tidyverse** provides abstract yet functional grammars to manipulate and visualize data in easier-to-comprehend form. One of the **tidyverse** packages, **dplyr** (Wickham et al., 2019b), showcases the value of a grammar as a principled vehicle to transform data for a wide range of data challenges, providing a consistent set of verbs: `mutate()`, `select()`, `filter()`, `summarize()`, and `arrange()`. Each verb focuses on a singular task. Most common data tasks can be rephrased and tackled with these five key verbs, in conjunction with `group_by()` to perform grouped operations.

The **tidyverse** largely formalizes exploratory data analysis. Many in the R community have adopted the **tidyverse** way of thinking and extended it to broader domains, such as simple features for spatial data in the **sf** package (Pebesma, 2018) and missing value handling in the **naniar** package (Tierney and Cook, 2018). This paper with the associated **tsibble** R package (Wang, Cook, and Hyndman, 2019b) extends the tidy way of thinking to temporal data.

For temporal data, the tidy definition needs additional criteria, that assist in handling the time context. This is addressed in the next section, and encompasses both time series and longitudinal data. It provides a unified framework to streamline the workflow from data preprocessing to visualization and modeling, as an integral part of a tidy data analysis.

index	key		measurements	

**Figure 3.2:** The architecture of the tsibble structure is built on top of the “tidy data” principles, with temporal semantics: index and key.

### 3.3 Contextual semantics

The choice of tidy representation of temporal data arises from a data- and model-oriented perspective, which can accommodate all of the operations that are to be performed on the data in time-based contexts. Figure 3.1 marks where this new abstraction is placed in the tidy model, which is referred to as a “tsibble”. The “tidy data” principles are adapted in tsibble with the following rules:

1. Index is a variable with inherent ordering from past to present.
2. Key is a set of variables that define observational units over time
3. Each observation should be uniquely identified by index and key.
4. Each observational unit should be measured at a common interval, if regularly spaced.

Figure 3.2 sketches out the data form required for a tsibble, an extension of the tidy format to the time domain. Beyond the layout, tsibble gives the contextual meaning to variables in order to construct the temporal data object, as newly introduced “index” and “key” semantics stated in definitions 1 and 2 above. Variables other than index and key are considered as measurements. Definitions 3 and 4 imply that a tsibble is *tidier* than tidy data, positioning itself as a model input that gives rise to more robust and reliable downstream analytics.

To materialize the abstraction of the tsibble, a subset of tuberculosis cases (World Health Organization, 2018), as presented in Table 3.1, is used as an example. It contains 12 observations and 5 variables landing in a tidy data form. Each observation comprises the number of people who are diagnosed with tuberculosis for each gender at three selected countries in the years of 2011 and 2012. From tidy data to tsibble data, index and key

should be declared: column `year` as the *index* variable, and column `country` together with `gender` as the *key* variables forming the observational units. Column `count` is the only measured variable in this data, but the data structure is sufficiently flexible to hold more measurements; for example, slotting the corresponding population size (if known) into the data column for normalizing the count later. Note, this data further satisfies the need for the distinct rows to be determined by index and key, and is regularly spaced over one-year intervals.

**Table 3.1:** *A small subset of estimates of tuberculosis burden collected by World Health Organization in 2011 and 2012, with 12 observations and 5 variables. The index refers to column `year`, the key to multiple columns: `country` and `gender`, and the measured variable to column `count`.*

country	continent	gender	year	count
Australia	Oceania	Female	2011	120
Australia	Oceania	Female	2012	125
Australia	Oceania	Male	2011	176
Australia	Oceania	Male	2012	161
New Zealand	Oceania	Female	2011	36
New Zealand	Oceania	Female	2012	23
New Zealand	Oceania	Male	2011	47
New Zealand	Oceania	Male	2012	42
United States of America	Americas	Female	2011	1170
United States of America	Americas	Female	2012	1158
United States of America	Americas	Male	2011	2489
United States of America	Americas	Male	2012	2380

The new tsibble structure bridges the gap between raw data and the rigorous state of temporal data analysis. The proposed contextual semantics is the new add-on to tidy data in order to support more intuitive time-related manipulations and enlighten new perspectives for time series and panel model inputs. Index, key and time interval form the three pillars to this new semantically structured temporal data. Each is now described in more detail.

### 3.3.1 Index

*Index is a variable with inherent ordering from past to present.*

Time provides the contextual basis for temporal data. Time can be seen in numerous representations, from sequential numerics to the most commonly accepted date-times. Regardless of this diversity, time should be inherently ordered from past to present, so should be the index variable to a tsibble.

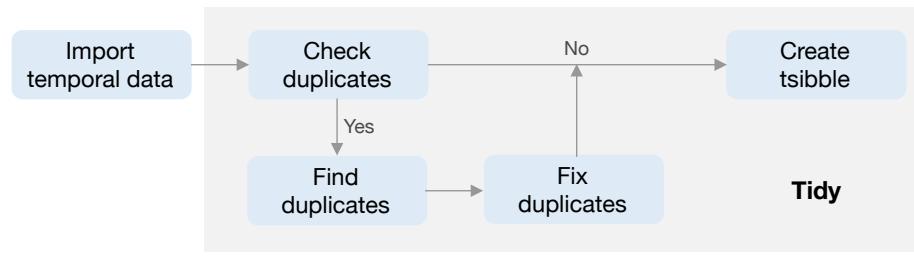
Index is an explicit data variable rather than a masked attribute (such as in the `ts` and `zoo` classes), exposing a need for more accessible and transparent time operations. It is often necessary to visualize and model seasonal effects of measurements of interest, meaning that time components, such as time of day and day of week, should be easily extracted from the index. When the index is available only as meta information, it creates an obstacle for analysts by complicating the writing of even simple queries, often requiring special purpose programming. From an analytical point of view this should be discouraged.

### 3.3.2 Key

*Key is a set of variables that define observational units over time.*

What subjects/entities are to be observed over time, leads to the second component of a tsibble–key. The key can consist of empty, single, or multiple variables identifying units measured along the way. When only a single observational unit is present in the table, no key needs to be specified. However, when multiple units exist in the data, the key should be supplied by identifying variables to sufficiently define the units. In longitudinal data, the key can be thought of as “panel” (such as in the Stata) but constrained to a single variable in existing data structures. In tsibble, the key allows for multiple variables of nesting, crossing, or union relations (Wilkinson, 2005), that can be useful for forecasting reconciliation (Hyndman and Athanasopoulos, 2017; Hyndman et al., 2018) and richer visualization. For example, Table 3.1 describes the number of tuberculosis cases for each gender across the countries every year. This suggests that the key comprises at least columns `gender` and `country`. Since `country` is nested within `continent`, `continent` can be included in the key specification, but is not compulsory.

*Each observation should be uniquely identified by index and key.*



**Figure 3.3:** Details about the tidy stage for a tsibble. Built on top of “tidy data”, each observation should be uniquely identified by index and key, thereby no duplicated key-index pairs.

Inspired by a “primary key” (Codd, 1970), a unique identifier for each observation in a relational database, the tsibble key also uniquely identifies each observational unit over time. When constructing a tsibble, any duplicates of key-index (i.e. key-value) pairs will fail, because duplicates signal a data quality issue, which would likely affect subsequent analyses and hence decision making. For example, either gender or country alone is not enough to be the key for the tuberculosis data. Analysts are encouraged to better understand the data, or reason about the process of data cleaning when handling duplicates. Figure 3.3 peels the tidy module with clear routes required for a tsibble. The rigidity of tsibble, as the fundamental data infrastructure, warrants the validity of temporal data analysis for the later stage.

Since observational units are embedded, modelling and forecasting across units and time in a tsibble will be simplified. The tsibble key plays a role of the central transit hub in connecting multiple tables managed by the data, models, and forecasts. This neatly decouples the expensive data copy from downstream summaries, significantly reducing the storage space.

### 3.3.3 Interval

*Each observational unit should be measured at a common interval, if regularly spaced.*

The principal divide of temporal data is regularly versus irregularly spaced data. Event data typically involves irregular time intervals, such as flight schedules or customer transactions. This type of data can flow into event-based data modeling, but would need to be processed, or regularized, to fit models that expect data with a fixed-time interval.

There are three possible interval types: fixed, unknown, and irregular. To determine the interval for regularly spaced data, tsibble computes the greatest common divisor as a fixed interval. If only one observation is available for each unit, which may occur after aggregating data, the interval is reported as unknown. When the data arrives with irregular time, like event data, the interval would be specified as irregular, to prevent the tsibble creator attempting to guess an interval.

To abide by the “tidy data” rules – “Each type of observational units should form a table” – in a tsibble each observational unit shares a common interval. This means that a tsibble will report one single interval, whether the data has a fixed or mixed set of intervals. To handle mixed interval data, it should be organized into separate tsibbles for a well-tailored analysis.

This tiny piece of information, the interval, is carried over for tsibble-centric operations. For example, this makes implicit missing time handling convenient, and harmoniously operates with statistical calculations, and models, on seasonal periods.

### 3.4 Temporal data pipelines

A data pipeline describes the flow of data through an analysis, and can generally assist in conceptualizing the process for a stream of problems. Mcilroy, Pinson, and Tague (1978) coined the term “pipelines” in software development while developing Unix at Bell Labs. In Unix-based computer operating systems, a pipeline chains together a series of operations based on their standard streams, so that the output of each program becomes the input to another. The Extract, Transform, and Load (ETL) process, described in recent data warehousing literature (Kimball and Caserta, 2011), outlines the workflow to prepare data for analysis, and can also be considered a data pipeline. Buja et al. (1988) describes a viewing pipeline for interactive statistical graphics, that takes control of the transformation from data to plot. Swayne, Cook, and Buja (1998), Swayne et al. (2003), Sutherland et al. (2000), Wickham et al. (2010) and Xie, Hofmann, and Cheng (2014) implemented data pipelines for the interactive statistical software **XGobi**, **GGobi**, **Orca**, **plumbr** and **cranvns**, respectively.

A fluent data pipeline anticipates a standard data structure. The tsibble data abstraction lays the plumbing for data analysis modules of transformation, visualization and modeling in temporal contexts. It provides a data infrastructure to a new ecosystem, **tidyverts**. (The name “tidyverts” is a play on the term “tidyverse” that acknowledges the time series analysis purpose.)

### 3.4.1 Transformation

The **tsibble** package not only provides a tsibble data object but also a domain specific language in R for transforming temporal data. It takes advantage of the wrangling verbs implemented in the **dplyr** package, and develops a suite of new tools for facilitating temporal manipulation for primarily easing two aspects: implicit missingness handlers and time-aware aggregations.

Implicit missingness are values that should be present but are absent. In regularly spaced temporal data, these are data entries that should be available at certain time points but are missing, leaving gaps in time. These can be detected when computing the interval estimate. It will be a problem for temporal models and operations like lag/lead are applied. A family of verbs are provided to help explore implicit missing values, and convert them into an explicit state, as follows:

- `has_gaps()` checks the existence of time gaps.
- `scan_gaps()` reveals all implicit missing observations.
- `count_gaps()` summarizes the time ranges that are absent from the data.
- `fill_gaps()` turns them into explicit ones, along with imputing by values or functions.

These verbs are evocative, and of simple interface. They, by default, look into gaps for each individual time period. Switching on the option `.full = TRUE` will fill in the full-length time span, and create fully balanced panels in longitudinal data, when possible.

The other important function, is an adverb, `index_by()`, which is the counterpart of `group_by()` in **dplyr**, grouping and partitioning by the index only. It is most often used in conjunction with `summarize()`, thus creating aggregations to higher-level time resolutions.

This combination automatically produces a new index and interval, and can also be used to regularize data of irregular interval.

In addition to the new verbs, the **dplyr** vocabulary has been adapted and expanded to facilitate temporal transformations. The **dplyr** suite showcases the general-purpose verbs for effectively manipulating tabular data. But these verbs need handling with care due to the context switch. A perceivable difference is summarizing variables between normal data and tsibble using `summarize()`. The former will reduce to a single summary, whereas the latter will obtain the index and their corresponding summaries.

Attention has been paid to warning and error handling. The principle that underpins most verbs is *a tsibble in and a tsibble out*, thereby striving to maintain a valid tsibble over the course of the transformation pipeline. If the desired temporal ordering is changed by row-wise verbs (such as `arrange()` and `slice()`), a warning is broadcast. If a tsibble cannot be maintained in the output of a pipeline module (likely occurring with column-wise verbs), for example the index is dropped by `select()-ing`, an error informs users of the problem and suggests alternatives. This avoids surprising users and reminds them of the time context. In general, users who are already familiar with the **tidyverse**, should have less resistance to learning the new semantics and verbs.

### 3.4.2 Visualization

The **ggplot2** package (Wickham, 2009) (as the implementation of grammar of graphics) builds a powerful graphical system to declaratively visualize data. The data underpinning of **ggplot2** is tidy data, and in turn tsibble integrates well with **ggplot2**. The integration encourages more flexible graphics for exploring temporal structures via index, and individual or group differences via key.

Line charts are universally accepted for ordered data, such as time series plots or spaghetti plots, depending on the fields. But they end up with exactly the same grammar: chronological time mapped to the horizontal axis, and the interested measurement on the vertical axis, for each unit. Many specialist plots centering around time series or longitudinal data, hence can be described and re-created under the umbrella of the grammar and **ggplot2**.

### 3.4.3 Model

Modeling is crucial to explanatory and predictive analytics, where time series and longitudinal data analysis diverge. The tsibble, as a model-oriented object, can flow into both types of modelling, and the new semantics (index and key) can be internally utilized to accelerate modelling.

Most time series models are univariate, such as ARIMA and Exponential Smoothing, modelling temporal dynamics for each series independently. The **fable** package (O'Hara-Wild, Hyndman, and Wang, 2019), currently under development, provides a tidy forecasting framework built on top of tsibble, with the goal of promoting transparent and human-centered forecasting practices. With the presence of the key, a tsibble can hold many series. Since models are fundamentally scalable, the `model()` and `forecast()` generics will take care of fitting and forecasting univariate models to each series across time in a tsibble at once.

Panel data models, however, put emphases on overall, within, and between variation both across individuals and time. Fixed and random effects models could be developed in line with the **fable** design.

### 3.4.4 Summary

To sum up, the tsibble abstraction provides a formal organization of forwarding tidy data to model-oriented temporal data. The supporting operations can be chained for sequencing analysis, articulating a data pipeline. As Friedman and Wand (2008) stated, “No matter how complex and polished the individual operations are, it is often the quality of the glue that most directly determines the power of the system.” A mini snippet below, illustrates how transformation and forecasting are glued together, to realize the fluent pipeline.

```
pedestrian %>%
  fill_gaps() %>% # turn implicit missingness to explicit
  filter(year(Date_Time) == 2016) %>% # subset data of year 2016
```

```
model(arima = ARIMA(Count)) %>% # fit ARIMA to each sensor  
forecast(h = days(2)) # forecast 2 days ahead
```

Here, the `pedestrian` dataset (City of Melbourne, 2017), available in the `tsibble` package is used. It contains hourly tallies of pedestrians at four counting sensors in 2015 and 2016 in inner Melbourne. The pipe operator `%>%` introduced in the `magrittr` package (Bache and Wickham, 2014) chains the verbs, read as “then”. A sequence of functions are composed in a way that can be naturally read from left to right, which improves the code readability. This code is read as “take the pedestrian data, fill the temporal gaps, filter to 2016 measurements, then apply an ARIMA model and forecast ahead 2 days.”

Piping coordinates a user’s analysis making it cleaner to follow, and permits a wider audience to follow the data analysis from code, without getting lost in a jungle of computational intricacies. It helps to (1) break up a big problem into more manageable blocks, (2) generate human readable analysis workflow, and (3) forestall introducing mistakes or, at least, make it possible to track, and fix, mistakes upstream through the pipeline.

## 3.5 Software structure and design decisions

The `tsibble` package development follows closely to the `tidyverse` design principles (Tidyverse Team, 2019).

### 3.5.1 Data first

The primary force that drives the software’s design choices is “data”. All functions in the package `tsibble` start with `data` or its variants as the first argument, namely “data first”. This lays out a consistent interface and addresses the significance of the data throughout the software.

Beyond the tools, the print display provides a quick and comprehensive glimpse of data in temporal contexts, particularly useful when handling a large collection of data. The contextual information provided by the `print()` function, shown below from Table 3.1,

contains (1) data dimension with its shorthand time interval, alongside time zone if date-times, (2) variables that constitute the “key” with the number of units. These summaries aid users in understanding their data better.

```
#> # A tsibble: 12 x 5 [1Y]
#> # Key:      country, gender [6]
#>   country    continent gender year count
#>   <chr>      <chr>     <chr>   <dbl> <dbl>
#> 1 Australia  Oceania   Female  2011   120
#> 2 Australia  Oceania   Female  2012   125
#> 3 Australia  Oceania   Male    2011   176
#> 4 Australia  Oceania   Male    2012   161
#> 5 New Zealand Oceania   Female  2011   36
#> # ... with 7 more rows
```

### 3.5.2 Functional programming

Rolling window calculations are widely used techniques in time series analysis, and often apply to other applications. These operations are dependent on having an ordering, particularly time ordering for temporal data. Three common types of variations for sliding window operations are:

1. **slide**: sliding window with overlapping observations.
2. **tile**: tiling window without overlapping observations.
3. **stretch**: fixing an initial window and expanding to include more observations.

Figure 3.4 shows animations of rolling windows for sliding, tiling and stretching on annual tuberculosis cases for Australia. A block of consecutive elements with a window size of 5 is initialized in each case, and the windows roll sequentially to the end of series, with average counts being computed within each window.

Rolling windows adapt to functional programming, for which the **purrr** package (Henry and Wickham, 2019a) sets a good example. These functions accept and return arbitrary

**Figure 3.4:** An illustration of a window of size 5 to compute rolling averages over annual tuberculosis cases in Australia using sliding, tiling and stretching. (Animation needs to be viewed with Adobe Acrobat Reader.)

inputs and outputs, with arbitrary methods. For example, moving averages anticipate numerics and produce averaged numerics via `mean()`. However, rolling window regression feeds a data frame into a linear regression method like `lm()`, and generates a complex object that contains coefficients, fitted values, and etc.

Rolling windows not only iterate but roll over a sequence of elements of a fixed window. A complete and consistent set of tools are available for facilitating window-related operations, a family of `slide()`, `tile()`, `stretch()`, and their variants. `slide()` expects one input, `slide2()` two inputs, and `pslide()` multiple inputs. For type stability, the functions always return lists. Other variants including `*_lgl()`, `*_int()`, `*_dbl()`, `*_chr()` return vectors of the corresponding types, as well as `*_dfr()` and `*_dfc()` for row-binding and column-binding data frames respectively. Their multiprocessing equivalents prefixed by `future_*`() enable rolling in parallel (Bengtsson, 2019; Vaughan and Dancho, 2018a).

### 3.5.3 Modularity

Modular programming is adopted in the design of the **tsibble** package. Modularity benefits users by providing small focused and cleaner chunks, and provides developers with simpler maintenance.

All user-facing functions can be roughly organized into three major chunks according to their functionality: vector functions (1d), table verbs (2d), and window family. Each chunk is an independent module, but works interdependently. Vector functions in the package mostly operate on time. The atomic functions (such as `yearmonth()` and `yearquarter()`) can be embedded in the `index_by()` verb to collapse a tsibble to a less granular interval. Since they are not tied to a tsibble, they can be used in a broader range of data applications not constrained to tsibble. On the other hand, the table verbs can incorporate many other vector functions from a third party, like the **lubridate** package.

### 3.5.4 Extensibility

As a fundamental infrastructure, extensibility is a design decision that was employed from the start of **tsibble**'s development. Contrary to the "data first" principle for end users, extensibility is developer focused and would be mostly used in dependent packages; it heavily relies on S3 classes and methods in R (Wickham, 2018). The package can be extended in two major ways: custom indexes and new tsibble classes.

Time representation could be arbitrary, for example R’s native `POSIXct` and `Date` for versatile date-times, nano time for nanosecond resolution in `nanotime` (Eddelbuettel and Silvestri, 2018), and numerics in simulation. Ordered factors can also be a source of time, such as month names, January to December, and weekdays, Monday to Sunday. The `tsibble` package supports an extensive range of index types from numerics to nano time, but there might be custom indexes used for some occasions, for example school semesters. These academic terms vary from one institution to another, with the academic year defined differently from a calendar year. A new index would be immediately recognized upon defining `index_valid()`, as long as it can be ordered from past to future. The interval regarding semesters is further outlined through `interval_pull()`. As a result, all `tsibble` methods such as `has_gaps()` and `fill_gaps()` will have instant support for data that contains this new index.

The class of `tsibble` is an underpinning for temporal data, and sub-classing a `tsibble` will be a demand. A low-level constructor `new_tsibble()` provides a vehicle to easily create a new subclass. This new object itself is a `tsibble`. It perhaps needs more metadata than those of a `tsibble`, that gives rise to a new data extension, for example prediction distributions to a forecasting `tsibble`.

### 3.5.5 Tidy evaluation

The `tsibble` packages leverages the `tidyverse` grammars and pipelines through tidy evaluation (Henry and Wickham, 2019c) via the `rlang` package (Henry and Wickham, 2019b). In particular, the table verbs extensively use tidy evaluation to evaluate computation in the context of `tsibble` data and spotlights the “tidy” interface that is compatible with the `tidyverse`. This not only saves a few keystrokes without explicitly repeating references to the data source, but the resulting code is typically cleaner and more expressive, when doing interactive data analysis.

## 3.6 Case studies

### 3.6.1 On-time performance for domestic flights in U.S.A

The dataset of on-time performance for US domestic flights in 2017 represents event-driven data caught in the wild, sourced from US Bureau of Transportation Statistics (Bureau of Transportation Statistics, 2018). It contains 5,548,445 operating flights with many measurements (such as departure delay, arrival delay in minutes, and other performance metrics) and detailed flight information (such as origin, destination, plane number and etc.) in a tabular format. This kind of event describes each flight scheduled for departure at a time point in its local time zone. Every single flight should be uniquely identified by the flight number and its scheduled departure time, from a passenger's point of view. In fact, it fails to pass the tsibble hurdle due to duplicates in the original data. An error is immediately raised when attempting to convert this data into a tsibble, and a closer inspection has to be carried out to locate the issue. The **tsibble** package provides tools to easily locate the duplicates in the data with `duplicates()`. The problematic entries are shown below.

```
#>   flight_num  sched_dep_datetime  sched_arr_datetime dep_delay arr_delay
#> 1      NK630  2017-08-03 17:45:00 2017-08-03 21:00:00       140      194
#> 2      NK630  2017-08-03 17:45:00 2017-08-03 21:00:00       140      194
#>   carrier tailnum origin dest air_time distance origin_city_name
#> 1      NK    N601NK    LAX   DEN        107       862    Los Angeles
#> 2      NK    N639NK    ORD   LGA        107       733      Chicago
#>   origin_state dest_city_name dest_state taxi_out taxi_in carrier_delay
#> 1          CA        Denver        CO        69       13            0
#> 2          IL      New York        NY        69       13            0
#>   weather_delay nas_delay security_delay late_aircraft_delay
#> 1            0        194            0            0
#> 2            0        194            0            0
```

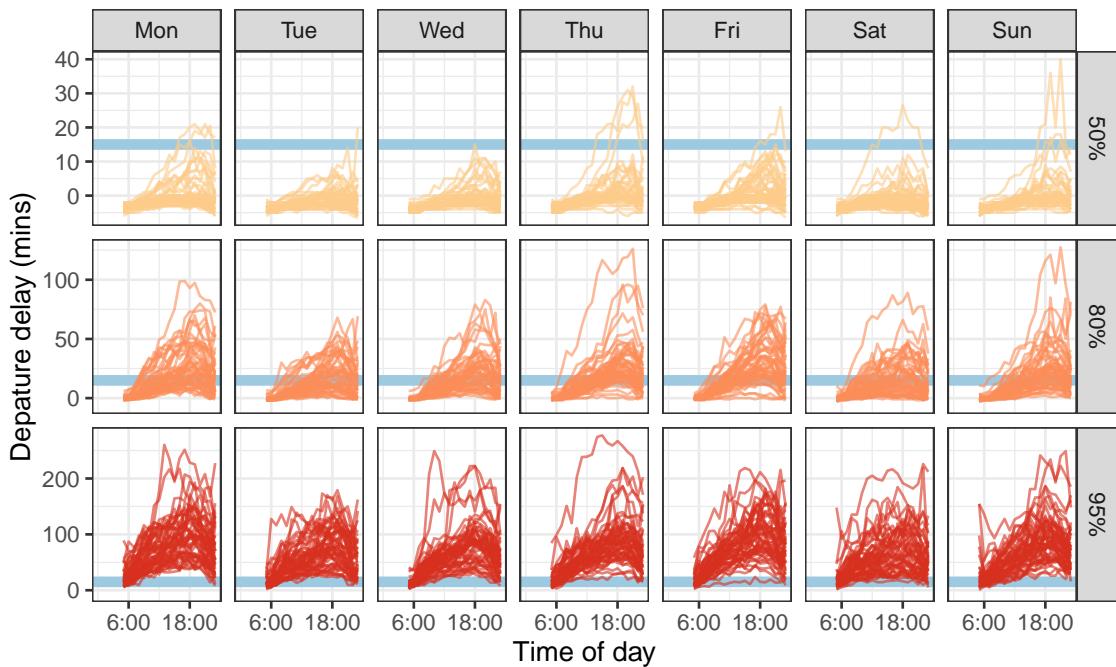
The issue was perhaps introduced when updating or entering the data into a system. The same flight is scheduled at exactly the same time, together with the same performance statistics but different flight details. As flight NK630 is usually scheduled at 17:45 from Chicago to New York (discovered by searching the full database), a decision is made to remove the first row from the duplicated entries before proceeding to the tsibble creation.

This dataset is intrinsically heterogeneous, encoded in numbers, strings, and date-times. The tsibble framework, as expected, incorporates this type of data without any loss of data richness and heterogeneity. To declare the flight data as a valid tsibble, column `sched_dep_datetime` is specified as the “index”, and column `flight_num` as the “key”. This data happens to be irregularly spaced, and hence switching to the irregular option is necessary. The software internally validates if the key and index produce distinct rows, and then sorts the key and the index from past to recent. When the tsibble creation is done, the print display is data-oriented and contextually informative, including dimensions, irregular interval with the time zone (`5,548,444 x 22 [!] <UTC>`) and the number of observational units (`flight_num [22,562]`).

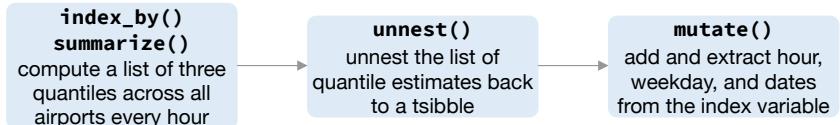
```
#> # A tsibble: 5,548,444 x 22 [!] <UTC>
#> # Key:           flight_num [22,562]
```

Transforming a tsibble for exploratory data analysis with a suite of time-aware and general-purpose manipulation verbs can result in well-constructed pipelines. A couple of use cases are described to show how to approach the interested questions by wrangling the tsibble while maintaining its temporal context.

What time of day and day of week should passengers travel to avoid suffering from horrible delay? Figure 3.5 plots hourly quantile estimates across day of week in the form of small multiples. The upper-tail delay behaviors are of primary interest, and hence 50%, 80% and 95% quantiles are computed. This pipeline is initialized by regularizing and reshaping the list of the upper quantiles of departure delays for each hour. To visualize the temporal profiles, the time components (for example hours and weekdays) are extracted from the index. The flow chart (Figure 3.6) demonstrates the operations undertaking in the data pipeline. The input to this pipeline is a tsibble of irregular interval for all flights,



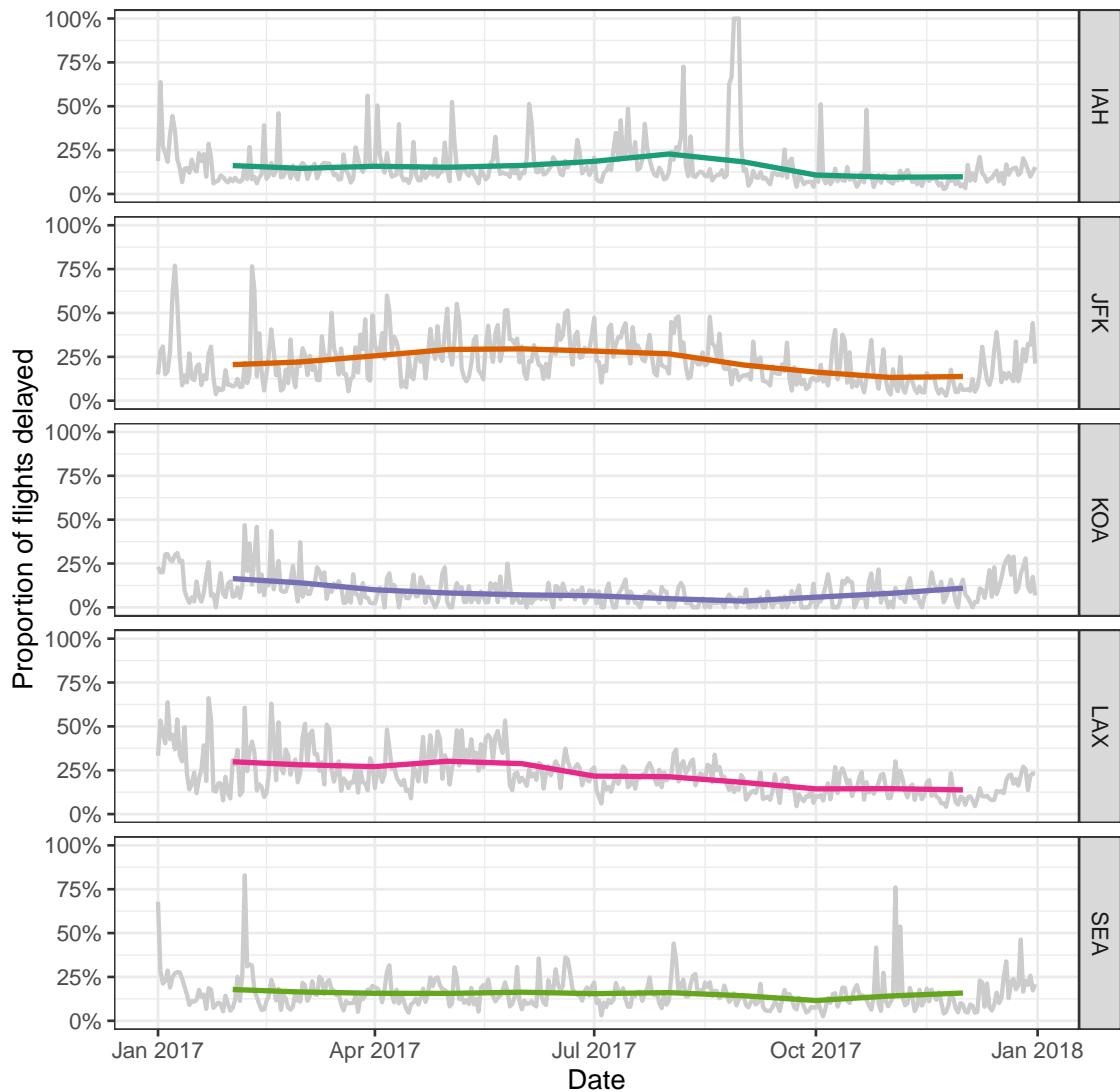
**Figure 3.5:** Small multiples of lines about departure delay against time of day, faceting day of week and 50%, 80% and 95% quantiles. A blue horizontal line indicates the 15-minute on-time standard to help grasp the delay severity. Passengers are apt to hold up around 18 during a day, and are recommended to travel early. The variations increase substantially as the upper tails.



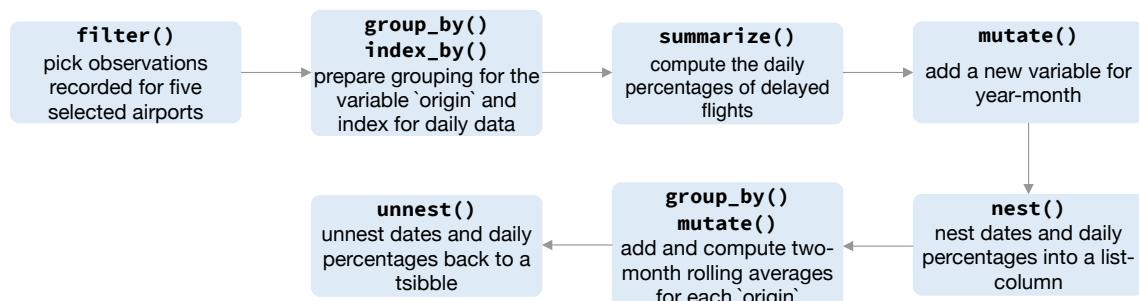
**Figure 3.6:** Flow chart illustrates the pipeline that preprocesses the data for creating Figure 3.5.

and the output ends up with a tsibble of one-hour interval by quantiles. To reduce the likelihood of suffering a delay, it is recommended to avoid the peak hour around 6pm (18) from Figure 3.5.

A closer examination of some big airports across the US will give an indication of how well the busiest airports manage the outflow traffic on a daily basis. A subset that contains observations for Houston (IAH), New York (JFK), Kalaoa (KOA), Los Angeles (LAX) and Seattle (SEA) airports is obtained first. The succeeding operations compute delayed percentages every day at each airport, which are shown as grey lines in Figure 3.7. Winter months tend to fluctuate a lot compared to the summer across all the airports. Superimposed on the plot are two-month moving averages, so the temporal trend is more



**Figure 3.7:** Daily delayed percentages for flight departure, with two-month moving averages overlaid, at five international airports. There are least fluctuations, and relatively fewer delays, observed at KOA airport.



**Figure 3.8:** Flow chart illustrating the pipeline that preprocessed the data for creating Figure 3.7.

visible. Since the number of days for each month is variable, moving averages over two months will require a weights input. But the weights specification can be avoided using a pair of commonly used rectangling verbs—`nest()` and `unnest()`, to wrap data frames partitioned by months into list-columns. The sliding operation with a large window size smooths out the fluctuations and gives a stable trend around 25% over the year, for IAH, JFK, LAX and SEA. LAX airport has seen a gradual decline in delays over the year, whereas the SEA airport has a steady delay. The IAH and JFK airports have more delays in the middle of year, while the KOA has the inverse pattern with higher delay percentage in both ends of the year. This pipeline gets the data into the daily series, and shifts the focus to five selected airports.

This case study begins with duplicates fixing, that resolved the issue for constructing the `tsibble`. A range of temporal transformations can be handled by many free-form combinations of verbs, facilitating exploratory visualization.

### 3.7 Smart-grid customer data in Australia

Sensors have been installed in households across major cities in Australia to collect data for the smart city project. One of the trials is monitoring households' electricity usage through installed smart meters in the area of Newcastle over 2010–2014 (Department of the Environment and Energy, 2018). Data from 2013 have been sliced to examine temporal patterns of customers' energy consumption with `tsibble` for this case study. Half-hourly general supply in kWh have been recorded for 2,924 customers in the data set, resulting in 46,102,229 observations in total. Daily high and low temperatures in Newcastle in 2013 provide explanatory variables other than time in a different data table (Bureau of Meteorology, 2019), obtained using the R package `bomrang` (Sparks et al., 2018). Aggregating the half-hourly energy data to the same daily time interval as the temperature data allows us to join the two data tables to explore how local weather can contribute to the variations of daily electricity use and the accuracy of demand forecasting.

During a power outage, electricity usage for some households may become unavailable, thus resulting in implicit missing values in the database. Gaps in time occur to 17.9% of the households in this dataset. It would be interesting to explore these missing patterns as

part of a preliminary analysis. Since the smart meters have been installed at different dates for each household, it is reasonable to assume that the records are obtainable for different time lengths for each household. Figure 3.9 shows the gaps for the top 49 households arranged in rows from high to low in tallies. (The remaining households values have been aggregated into a single batch and appear at the top.) Missing values can be seen to occur at any time during the entire span. A small number of customers have undergone energy unavailability in consecutive hours, indicated by a line range in the plot. On the other hand, the majority suffer occasional outages with more frequent occurrence in January.

Aggregation across all individuals helps to sketch a big picture of the behavioral change over time in the region, organized into a calendar display (Figure 3.10) using the **sugrrants** package (Wang, Cook, and Hyndman, 2018). Each glyph represents the daily pattern of average residential electricity usage every thirty minutes. Higher consumption is indicated by higher values, and typically occurs in daylight hours. Color indicates hot days. The daily snapshots vary depending on the season in the year. During the summer months (December and January), the late-afternoon peak becomes the dominant usage pattern. This is probably driven by the use of air conditioning, because high peaks mostly correspond to hot days, where daily average temperatures are greater than 25 degrees Celsius. In the winter time (July and August) the daily pattern sees two peaks, which is probably due to heating in the morning and evening.

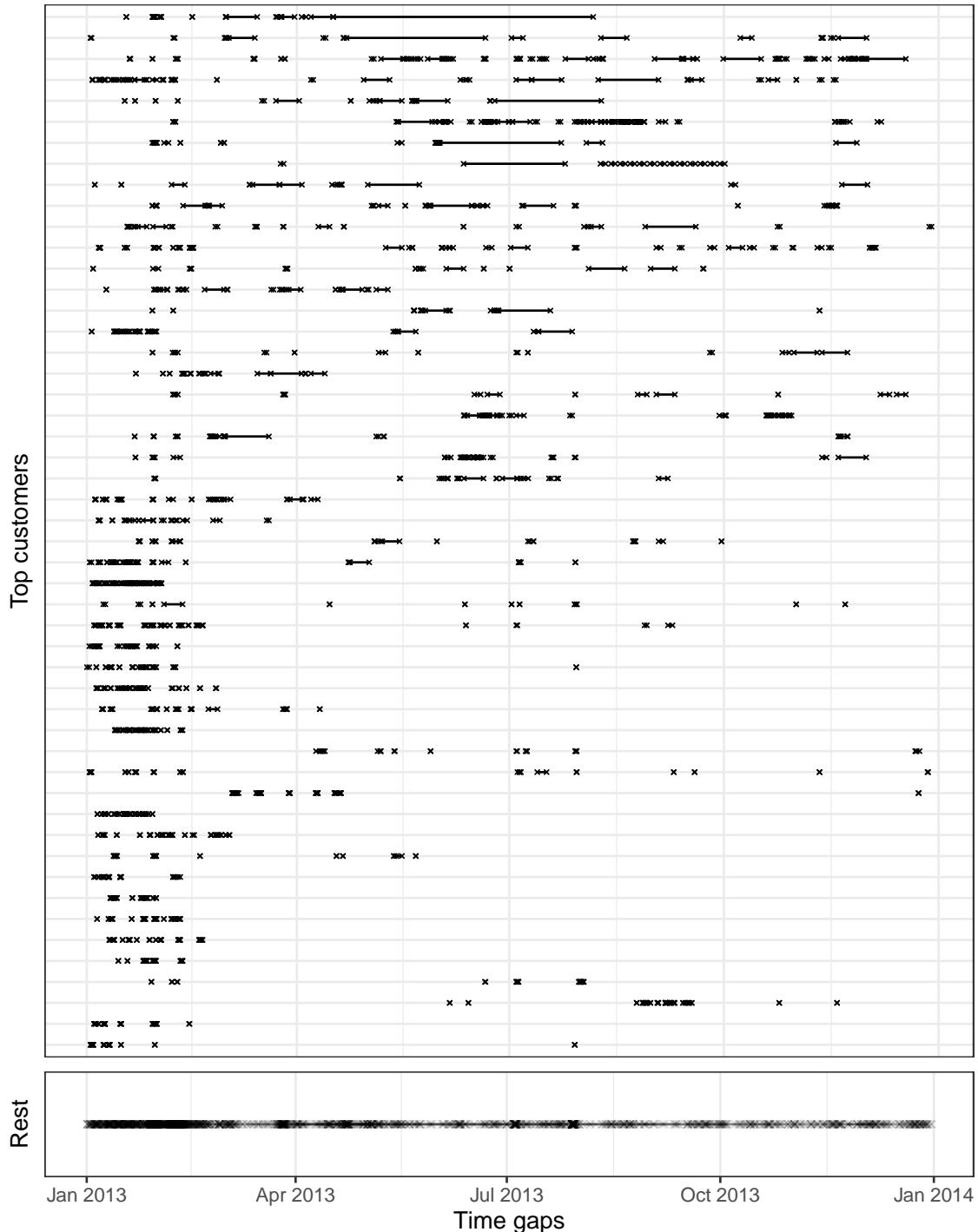
A common practice with energy data analysis is load forecasting, because providers need to know they have capacity to supply electricity. To illustrate the pipeline including modeling, here demand is modeled for December 2013, with the usage forecast for the last day (48 steps ahead because the data is half-hourly). The energy data for the last day is not used for modeling. ARIMA models with and without a temperature covariate are fitted using automatic order selection (Hyndman and Khandakar, 2008). The logarithmic transformation is applied to the average demand to ensure positive forecasts. Figure 3.11 plots one-day forecasts from both models against the actual demand, for the last two-week window. The ARIMA model which includes the average temperature covariate gives a better fit than the one without, although both tend to underestimate the night demand.

The forecasting performance is reported in Table 3.2, consistent with the findings in Figure 3.11.

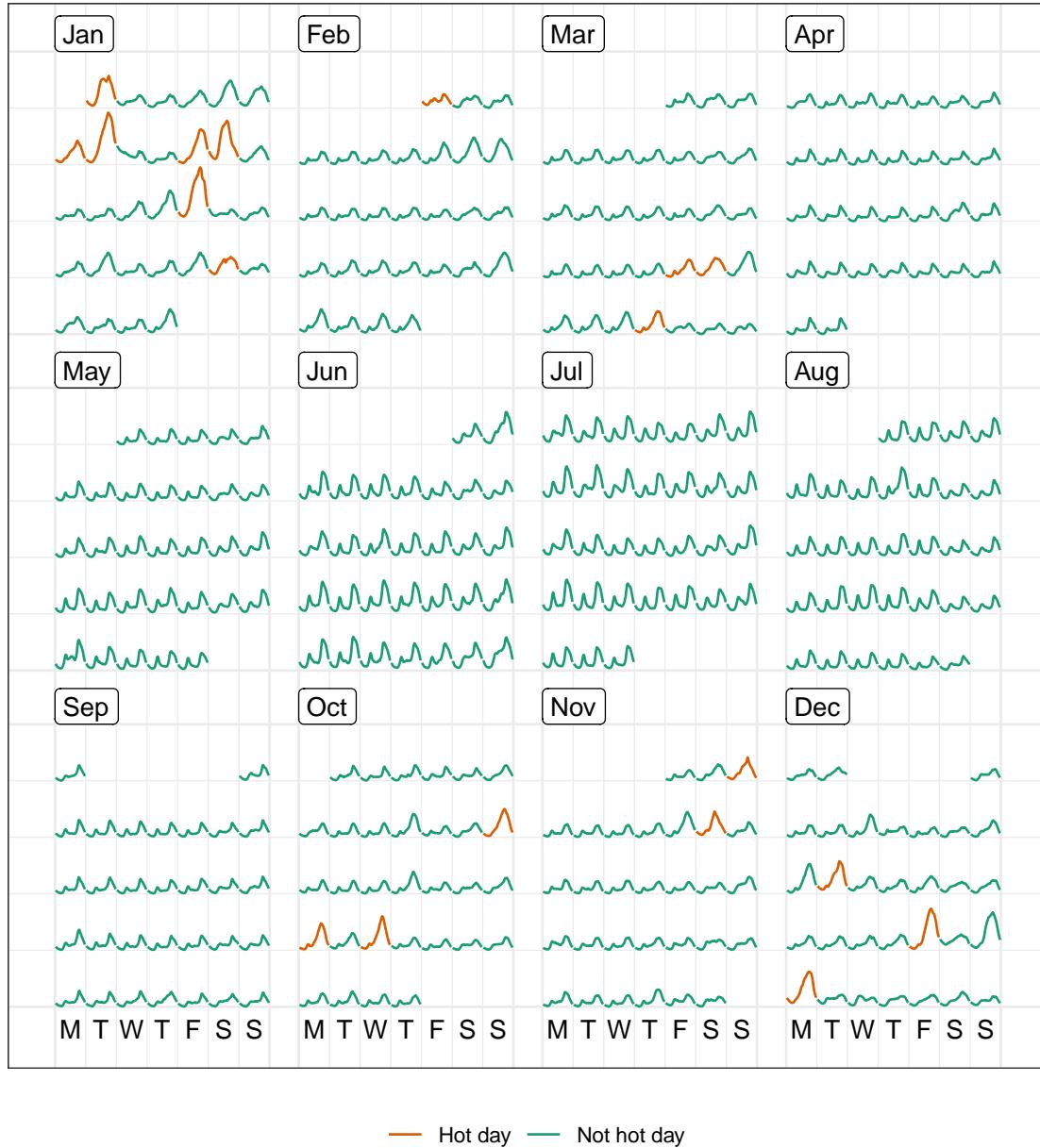
**Table 3.2:** Accuracy measures to evaluate the forecasting performance between ARIMA models with and without temperatures, using the validation set.

model	ME	RMSE	MAE	MPE	MAPE
temperature	-0.009	0.030	0.025	-6.782	11.446
w/o temperature	0.016	0.043	0.032	2.634	12.599

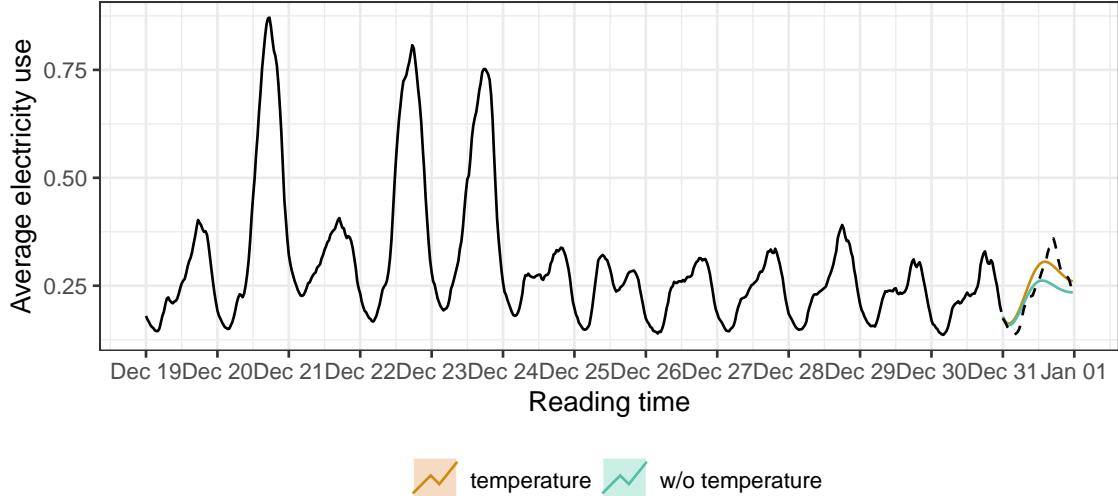
This case study demonstrates the significance of tsibble in lubricating the plumbing of handling time gaps, visualizing, and forecasting in general.



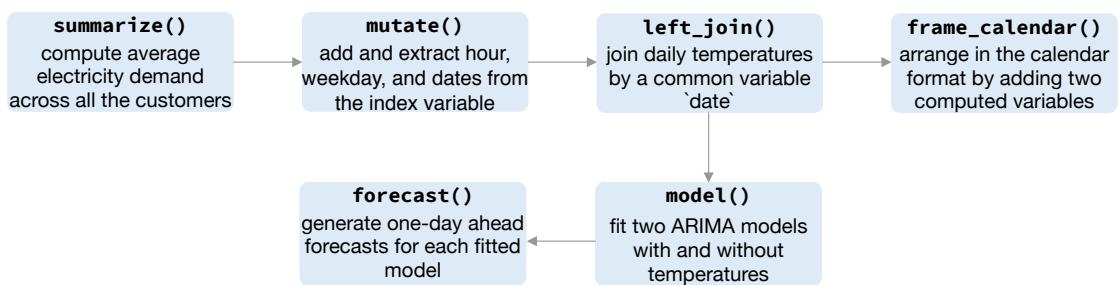
**Figure 3.9:** Exploring temporal location of missing values, using time gap plots for the 49 customers with most implicit missing values. The remaining customers are grouped into the one line in the bottom panel. Each cross represents an observation missing in time and a line between two dots shows continuous missingness over time. Missing values tend to occur at various times, although there is a higher concentration of missing in January and February for most customers.



**Figure 3.10:** Half-hourly average electricity use across all customers in the region, organized into calendar format, with color indicating hot days. Energy use of hot days tends to be higher, suggesting air conditioner use. Days in the winter months have a double peak suggesting morning and evening heater use.



**Figure 3.11:** One-day (48 steps ahead) forecasts generated by ARIMA models, with and without a temperature covariate, plotted against the actual demand. Both nicely capture the temporal dynamics, but ARIMA with temperature performs better than the model without.



**Figure 3.12:** Flow chart illustrating the pipeline involved for creating Figure 3.10 and Figure 3.11.

### 3.8 Conclusion and future work

The data abstraction, `tsibble`, for representing temporal data, extends the tidy data principles into the time domain. Tidy data takes shape in the realm of time with the new contextual semantics: index and key. The index variable provides direct support to an exhaustive set of ordered objects. The key, which can consist of single or multiple variables, identifies observational units over time. These semantics further determine unique data entries required for a valid `tsibble`. It shepherds raw temporal data through the tidying stage of an analysis pipeline to the next exploration stage to fluently gain insights.

The supporting toolkits articulate the temporal data pipeline, with the shared goal of reducing the time between framing of data questions and the code realization. The rapid iteration for broader understanding of the data is achieved through frictionlessly shifting among transformation, visualization, and modelling, using the standardized `tsibble` data infrastructure.

Future work includes allowing user-defined calendars, so that the `tsibble` structure respects structural missing observations. For example, a call center may operate only between 9:00 am and 5:00 pm on week days, and stock trading resumes on Monday straight after Friday. No data available outside trading hours would be labeled as structural missingness. Customer calendars can be embedded into the `tsibble` framework in theory. A few R packages provide functionality to create and manage many specific calendars, such as the `bizdays` package (Freitas, 2018) for business days calendars. However, a generic flexible calendar system is lacking, and requires complex implementation, so this is left for future work.

### Acknowledgments

The authors would like to thank Mitchell O'Hara-Wild for many discussions on the software development and Davis Vaughan for contributing ideas on rolling window functions. We also thank Stuart Lee for the feedback on this manuscript. We are grateful for anonymous reviewers for helpful feedback that has led to many improvements in the paper. This article was created with knitr (Xie, 2015) and R Markdown (Xie, Allaire,

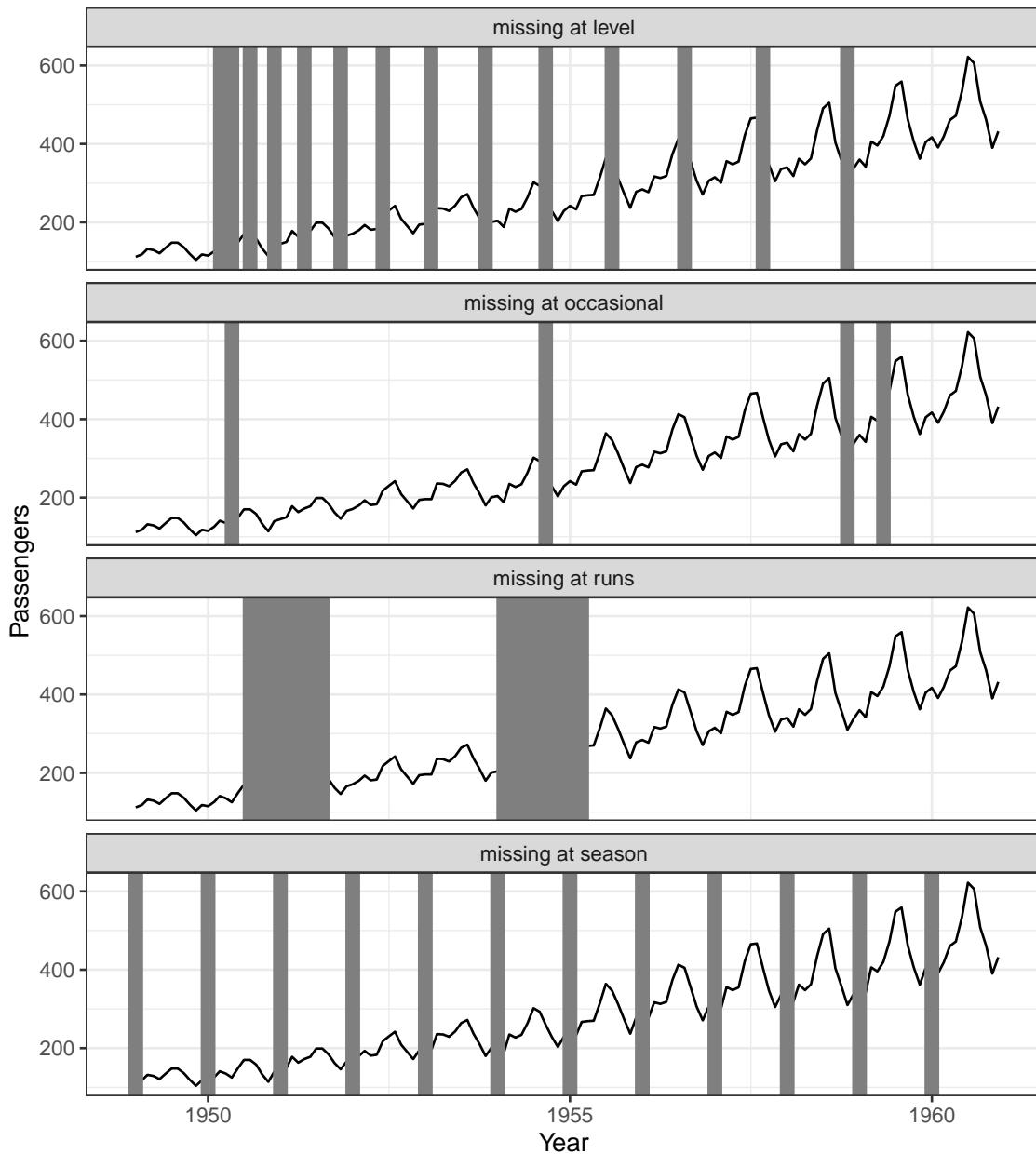
and Grolemund, 2018). The project's Github repository <https://github.com/earowang/paper-tsibble> houses all materials required to reproduce this article and a history of the changes.

## **Chapter 4**

# **Data representation, visual and analytical techniques for demystifying temporal missing data**

Missing data provokes an air of mystery, that makes analysts itching throughout the exploration loop of transformation, visualization, and modelling. How to handle missing values involves decisions with many degrees of freedom, lending itself to a tedious and unwieldy process. The challenge of missingness roots in seeing what isn't there. The aim of this work is to clear that mysterious air away from missing data with the focus of temporal contexts from the data-centric perspective. A new sparse representation facilitates to index the runs of missings in time efficiently, with supporting operations and visual methods. This places missing data solely in the spotlight, speaking for themselves. When too many missings are scattered across variables and observations over time, missing data polishing strategies are populated and formulated. This equips analysts with tidy tools to iteratively remove missings from rows and columns, while keeping the temporal nature intact. The accompanying software is the R package **mists**

### **4.1 Introduction**



**Figure 4.1:** Time series lines along with missing distributions plotted for monthly number of international airline passengers from 1949 to 1960. This series shows an upward trend with strong seasonal patterns. The gray blocks indicate the missingness. To generate missing data in time, the process could be classified into four types of temporal missings: (1) missing at level, (2) missing at occasional, (3) missing at runs, (4) missing at season.

# **Chapter 5**

## **Conclusion and future plans**

The three papers assembled in this thesis, share a common theme of exploratory analysis for temporal data using tidy tools. Chapter 2, “Calendar-based graphics for visualizing people’s daily schedules”, described a new calendar-based display. Chapter 3, “A new tidy data structure to support exploration and modeling of temporal data”, proposed a new temporal data abstraction. Chapter 4, “”, demystified missing data in time. These papers are bundled with software. In this conclusion, I will briefly summarise each package and their impact, and discuss the future directions of my research.

### **5.1 Software development**

A particular emphasis of this thesis is translating research methodologies in the form of open source R packages: **sugrrants**, **tsibble**, and **mists**.

#### **5.1.1 sugrrants**

The **sugrrants** package implements the idea of displaying data in the familiar calendar style in the `frame_calendar()` and `facet_calendar()`. The research article, a shorter version of Chapter 2, has been awarded the best student paper prize from ASA Sections on Statistical Computing and Statistical Graphics and ACEMS Business Analytics in 2018. There have been a grand total of 14,347 downloads from the RStudio mirror (one of 90 CRAN mirrors) dating from 2017-07-28 to 2019-09-02; and it has been starred 47 times

on Github so far. The homepage at <https://pkg.earo.me/sugrrants> contains detailed documentation and a vignette on `frame_calendar()`.

### 5.1.2 **tsibble**

The **tsibble** package provides a data infrastructure and a domain specific language in R for representing and manipulating tidy temporal data. This package is aimed for the fundamental architecture that other temporal tools will be built upon. For example, a new suite of time series analysis packages, titled “[tidyverts](#)”, have been developed for the new “`tsibble`” object. The **tsibble** package has won 2019 John Chambers Statistical Software Award from ASA Sections on Statistical Computing and Statistical Graphics. It has been downloaded 37,684 times from the RStudio mirror since it landed on CRAN on 2018-01-09; and it has received 234 stars on Github. These metrics are the indicators of my research impact, the recognition by professionals, and the uptakes by users. The website (<https://tsibble.tidyverts.org>) includes full documentation and three vignettes about the package usage.

### 5.1.3 **mists**

The **mists** package aims at exploring missing values for temporal data both analytically and graphically. It implements a compact abstraction for efficiently indexing missing data in time, along with numerical and visual methods. It also brings missing data polishing techniques into live. The Github repository has received 22 stars, but the package is not on CRAN yet. The documentation site is available at <https://pkg.earo.me/mists>.

## 5.2 Future work

### 5.2.1 Process for generating missing data in time

Missing values in cross-sectional data are typically characterised by the overall, row-wise, and column-wise missings. Neither, however, captures the dynamics in temporal data. How to characterise temporal missingness receives little research. A good metric of such

could possibly shed lights on the process for generating and imputing missing data in time.

I will expand on Chapter 4, to develop systematic missing data generating processes. Generating temporal missingness can be decomposed into two steps: (1) to inject point missings along indices and (2) to simulate the corresponding lengths.

### **5.2.2 Visual methods for temporal data of nesting and crossing interactions**

A collection of time series are often structured in a way that allows nesting and crossing interactions (Hyndman and Athanasopoulos, 2017). For example, a manufacturing company can add up every store's sales by region, by state and by country, which gives a strictly hierarchical time series; alternatively, they can gather the sales based on common attributes such as store, brand, price range and so forth, which leads to a crossed configuration. Nesting is a special case of crossing, with parent-children relations involved. Temporal information such as date-times is also intrinsically hierarchical, seconds nested within minutes, hours, and etc. The new tsibble structure has the neat capability of supporting these structural embeddings.

Numerous nesting and crossing combinations can yield unwieldy plots, in many of which an abundance of information are possibly buried. Focus-plus-context visualisation with interactivity comes to rescue. Dual contexts, structurally informative subjects and time, provide the source and visual clues for elegant navigation. Interactions on contextual plots control what to be visualised in the main plots, that form our focuses. Many kinds of visual displays can be generated to progressively build a richer data picture through guided or self explorations.



# Bibliography

- Bache, SM and H Wickham (2014). *magrittr: A Forward-Pipe Operator for R.* R package version 1.5. <https://CRAN.R-project.org/package=magrittr>.
- Becker, RA, WS Cleveland, and MJ Shyu (1996). The Visual Design and Control of Trellis Display. en. *Journal of Computational and Graphical Statistics* 5(2), 123–155. (Visited on 04/23/2019).
- Bengtsson, H (2019). *future: Unified Parallel and Distributed Processing in R for Everyone.* R package version 1.11.1.1. <https://CRAN.R-project.org/package=future>.
- Buja, A, D Asimov, C Hurley, and JA McDonald (1988). “Elements of a Viewing Pipeline for Data Analysis”. In: *Dynamic Graphics for Statistics*. Ed. by WS Cleveland and ME McGill. Belmont, California: Wadsworth, Inc.
- Bureau of Meteorology (2019). *Australia's National Weather Data*. Australian Government, Bureau of Meteorology. <https://data.gov.au/dataset/4e21dea3-9b87-4610-94c7-15a8a77907ef> (visited on 01/12/2019).
- Bureau of Transportation Statistics (2018). *Carrier On-Time Performance*. 1200 New Jersey Avenue, SE Washington, DC 20590. [https://www.transtats.bts.gov/DL\\_SelectFields.asp?Table\\_ID=236](https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236) (visited on 09/26/2018).
- Chambers, JM, WS Cleveland, B Kleiner, and PA Tukey (2017). *Graphical Methods for Data Analysis*. New York, NY: Chapman and Hall/CRC.
- Chang, W, J Cheng, J Allaire, Y Xie, and J McPherson (2019). *shiny: Web Application Framework for R.* R package version 1.3.2. <https://CRAN.R-project.org/package=shiny>.
- City of Melbourne (2017). *Pedestrian Volume in Melbourne*. Last accessed 2017-07-09. <http://www.pedestrian.melbourne.vic.gov.au>.

## BIBLIOGRAPHY

---

- Cleveland, WS (1979). Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association* **74**(368), 829–836.
- Cleveland, WS and R McGill (1984). Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association* **79**(387), 531–554.
- Codd, EF (1970). A relational model of data for large shared data banks. *Communications of the ACM* **13**(6), 377–387.
- Croissant, Y and G Millo (2008). Panel Data Econometrics in R: The plm Package. *Journal of Statistical Software, Articles* **27**(2), 1–43.
- Department of the Environment and Energy (2018). *Smart-Grid Smart-City Customer Trial Data*. Australian Government, Department of the Environment and Energy. <https://data.gov.au/dataset/4e21dea3-9b87-4610-94c7-15a8a77907ef> (visited on 11/19/2018).
- Eddelbuettel, D and L Silvestri (2018). *nanotime: Nanosecond-Resolution Time for R*. R package version 0.2.3. <https://CRAN.R-project.org/package=nanotime>.
- Freitas, W (2018). *bizdays: Business Days Calculations and Utilities*. R package version 1.0.6. <https://CRAN.R-project.org/package=bizdays>.
- Friedman, DP and M Wand (2008). *Essentials of Programming Languages, 3rd Edition*. 3rd ed. The MIT Press.
- Hafen, R (2019). *geofacet: 'ggplot2' Faceting Utilities for Geographical Data*. R package version 0.1.10. <https://CRAN.R-project.org/package=geofacet>.
- Henry, L and H Wickham (2019a). *purrr: Functional Programming Tools*. R package version 0.3.0. <https://CRAN.R-project.org/package=purrr>.
- Henry, L and H Wickham (2019b). *rlang: Functions for Base Types and Core R and 'Tidyverse' Features*. R package version 0.3.1. <https://CRAN.R-project.org/package=rlang>.
- Henry, L and H Wickham (2019c). *Tidy Tidyverse Design Principles*. <https://principles.tidyverse.org>.
- Hyndman, RJ and G Athanasopoulos (2017). *Forecasting: Principles and Practice*. Melbourne, Australia: OTexts. [OTexts.org/fpp2](http://OTexts.org/fpp2).
- Hyndman, R and Y Khandakar (2008). Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software, Articles* **27**(3), 1–22.

- Hyndman, R, A Lee, E Wang, and S Wickramasuriya (2018). *hts: Hierarchical and Grouped Time Series*. R package version 5.1.5. <https://CRAN.R-project.org/package=hts>.
- Jacobs, J (2017). *ggtcal: Calendar Plot Using 'ggplot2'*. R package version 0.1.0. <https://github.com/jayjacobs/ggtcal>.
- Jones, H (2016). *Calendar Heatmap*. Last accessed 2019-02-28. <https://rpubs.com/haj3/calheatmap>.
- Kimball, R and J Caserta (2011). *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons.
- Kothari, A and Ather (2016). *ggTimeSeries: Nicer Time Series Visualisations with ggplot syntax*. R package version 0.1. <https://github.com/Ather-Energy/ggTimeSeries>.
- Lam, H, T Munzner, and R Kincaid (2007). Overview Use in Multiple Visual Information Resolution Interfaces. *IEEE Transactions on Visualization and Computer Graphics* **13**(6), 1278–1285.
- Long, JA (2019). *panelr: Regression Models and Utilities for Repeated Measures and Panel Data*. R package version 0.7.1. <https://CRAN.R-project.org/package=panelr>.
- Mcilroy, D, E Pinson, and B Tague (1978). Unix Time-Sharing System Forward. *The Bell System Technical Journal*, 1902–1903.
- O'Hara-Wild, M, R Hyndman, and E Wang (2019). *fable: Forecasting Models for Tidy Time Series*. R package version 0.1.0. <https://fable.tidyverts.org>.
- Pebesma, E (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal* **10**(1), 439–446.
- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. <https://www.R-project.org/>.
- Ryan, JA and JM Ulrich (2018). *xts: eXtensible Time Series*. R package version 0.11-0. <https://CRAN.R-project.org/package=xts>.
- SAS Institute Inc. (2018). *SAS/STAT Software, Version 9.4*. Cary, NC. <http://www.sas.com>.
- Sievert, C (2018). *plotly for R*. Last accessed 2018-10-20. <http://plotly-r.com>.
- Sparks, A, J Carroll, D Marchiori, M Padgham, H Parsonage, and K Pembleton (2018). *bomrang: Australian Government Bureau of Meteorology (BOM) Data from R*. R package version 0.4.0. <https://CRAN.R-project.org/package=bomrang>.

## BIBLIOGRAPHY

---

- StataCorp (2017). *Stata Statistical Software: Release 15*. StataCorp LLC. College Station, TX, United States. <https://www.stata.com>.
- Sutherland, P, A Rossini, T Lumley, N Lewin-Koh, J Dickerson, Z Cox, and D Cook (2000). Orca: A Visualization Toolkit for High-Dimensional Data. *Journal of Computational and Graphical Statistics* **9**(3), 509–529.
- Swayne, DF, D Cook, and A Buja (1998). XGobi: Interactive Dynamic Data Visualization in the X Window System. *Journal of Computational and Graphical Statistics* **7**(1), 113–130.
- Swayne, DF, D Temple Lang, A Buja, and D Cook (2003). GGobi: evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis* **43**, 423–444.
- Tidyverse Team (2019). *Tidy Evaluation*. <https://tidyeval.tidyverse.org>.
- Tierney, NJ and D Cook (2018). *Expanding tidy data principles to facilitate missing data exploration, visualization and assessment of imputations*. eprint: [arXiv:1809.02264](https://arxiv.org/abs/1809.02264).
- Tufte, ER (1983). *The Visual Display of Quantitative Information*. Graphics press Cheshire, CT.
- Unwin, A and P Valero-Mora (2018). Ensemble Graphics. en. *Journal of Computational and Graphical Statistics* **27**(1), 157–165. (Visited on 04/23/2019).
- Van Wijk, JJ and ER Van Selow (1999). Cluster and Calendar Based Visualization of Time Series Data. In: *Information Visualization, 1999. INFOVIS 1999 Proceedings. IEEE Symposium on*. IEEE, pp.4–9.
- Vaughan, D and M Dancho (2018a). *furrr: Apply Mapping Functions in Parallel using Futures*. R package version 0.1.0. <https://CRAN.R-project.org/package=furrr>.
- Vaughan, D and M Dancho (2018b). *tibbletime: Time Aware Tibbles*. R package version 0.1.1. <https://CRAN.R-project.org/package=tibbletime>.
- Wang, E (2019). *wanderer4melb: Shiny App for Wandering Around the Downtown Melbourne 2016*. R package version 0.2.0. <https://github.com/earowang/wanderer4melb>.
- Wang, E, D Cook, and R Hyndman (2019a). *sugrrants: Supporting Graphs for Analysing Time Series*. R package version 0.2.4. <https://CRAN.R-project.org/package=sugrrants>.
- Wang, E, D Cook, and R Hyndman (2019b). *tsibble: Tidy Temporal Data Frames and Tools*. R package version 0.8.3. <https://tsibble.tidyverts.org>.

- Wang, E, D Cook, and RJ Hyndman (2018). *Calendar-based graphics for visualizing people's daily schedules*. eprint: [arXiv:1810.09624](https://arxiv.org/abs/1810.09624).
- Wickham, H (2009). *ggplot2: Elegant Graphics for Data Analysis*. New York, NY: Springer-Verlag New York.
- Wickham, H (2014). Tidy Data. *Journal of Statistical Software* **59**(10), 1–23.
- Wickham, H (2017). *tidyverse: Easily Install and Load the 'Tidyverse'*. R package version 1.2.1. <https://CRAN.R-project.org/package=tidyverse>.
- Wickham, H (2018). *Advanced R*. 2nd ed. Chapman & Hall. <https://adv-r.hadley.nz/>.
- Wickham, H, W Chang, L Henry, TL Pedersen, K Takahashi, C Wilke, and K Woo (2019a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.1. <https://CRAN.R-project.org/package=ggplot2>.
- Wickham, H, R François, L Henry, and K Müller (2019b). *dplyr: A Grammar of Data Manipulation*. <http://dplyr.tidyverse.org>, <https://github.com/tidyverse/dplyr>.
- Wickham, H and G Grolemund (2016). *R for Data Science*. O'Reilly Media. <http://r4ds.had.co.nz/>.
- Wickham, H, H Hofmann, C Wickham, and D Cook (2012). Glyph-maps for Visually Exploring Temporal Patterns in Climate Data and Models. *Environmetrics* **23**(5), 382–393.
- Wickham, H, M Lawrence, D Cook, A Buja, H Hofmann, and DF Swayne (2010). The Plumbing of Interactive Graphics. *Computational Statistics*, 1–7.
- Wilkinson, L (2005). *The Grammar of Graphics (Statistics and Computing)*. Secaucus, NJ: Springer-Verlag New York, Inc.
- Wong, J (2013). *TimeProjection: Time Projections*. R package version 0.2.0. <https://CRAN.R-project.org/package=TimeProjection>.
- World Health Organization (2018). *Tuberculosis Data*. Block 3510, Jalan Teknokrat 6, 63000 Cyberjaya, Selangor, Malaysia. <http://www.who.int/tb/country/data/download/en/> (visited on 06/05/2018).
- Xie, Y (2015). *Dynamic Documents with R and knitr*. 2nd. Boca Raton, Florida: Chapman and Hall/CRC. <https://yihui.name/knitr/>.

## BIBLIOGRAPHY

---

- Xie, Y (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. ISBN 978-1138700109. Boca Raton, Florida: Chapman and Hall/CRC. <https://github.com/rstudio/bookdown>.
- Xie, Y, J Allaire, and G Grolemund (2018). *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman and Hall/CRC. <https://bookdown.org/yihui/rmarkdown>.
- Xie, Y, H Hofmann, and X Cheng (2014). Reactive Programming for Interactive Graphics. *Statistical Science* **29**(2), 201–213.
- Zeileis, A and G Grothendieck (2005). zoo: S3 Infrastructure for Regular and Irregular Time Series. *Journal of Statistical Software* **14**(6), 1–27.