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

March 2019

Contents

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 requested by the *Journal of Computational and Graphical Statistics* for revision and resubmission. It has won the 2018 ASA Statistical Graphics Student Paper Award and the 2018 ACEMS Business Analytics Prize. Chapter 3 has been submitted to *Journal of Computational and Graphical Statistics* and is currently under review. The accompanying R package **tsibble** has won the 2019 John Chambers Statistical Software Award. 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

Chapter 1

Introduction

This thesis contributes to three different facets of working with temporal data, and is grounded in the exploratory data analysis pipeline for time series. It begins the pipeline by visualising data in the time context: Section 1.1 introduces the calendar-based grammar of graphics for plotting people's daily activities in a calendar layout, detailed in Chapter 2. This in turn motivates a new data abstraction in need of streamlining transformation, visualisation, and modelling for analysing time series: Section 1.2 introduces the "tsibble" data structure to form the foundation of time series pipelines, expanded in Chapter 3. The tsibble representation breeds the ground for framing missing value of time series in the data-centric workflow: Section 1.3 introduces explanatory tools for understanding missing patterns in time, described in Chapter 4. Finally, Chapter 5 summarises the software tools developed for the work and the impact, and discusses the future plans.

It has been a long-held belief that exploratory data analysis is a highly ad hoc statistical area, impossible to teach or formalise. This has radically changed in recent years: by working with the "tidy data" (Wickham, 2014) as a fundamental unit, data plots and data wrangling can be formally described using a generalised set of abstract grammar, for example the grammar of graphics and data manipulation, as implemented in the **ggplot2** (Wickham et al., 2018a) and **dplyr** (Wickham et al., 2018b) R packages. These are the core of the **tidyverse** concepts and suite of tools. My work leverages the tidyverse way of

thinking to time series analysis, by providing better tools for developing data fluency in time series.

1.1 Calendar-based graphics

A grammar of graphics was first proposed by Wilkinson (2005) and extended by Wickham (2009). These methods establish a conceptual framework for mapping data to graphical form. It is the analogue to thinking of statistics as a mapping of random variables. For example, a mean is the mapping of n independent and identically distributed random variables, $X_1, \ldots, X_n$, $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$. With this functional mapping, the statistic can be computed on a sample, and the properties of the statistic can be analytically studied.

The grammar formulates a set of rules to map variables to visual quantities. Therefore, we can make the comparisons between different displays and describe the differences under a standard framework. The grammar enables one to understand the process of creating visual graphics and the data structure behind the statistical plots. It is efficient and replicable to make any type of graphic display. The R package **ggplot2** (Wickham et al., 2018a) is an implementation of the grammar of graphics.

A new calendar-based graphics have been created and implemented to better visualise sub-daily data related to human behaviours. Tidy temporal data builds the foundation of calendarising the data using the data re-structuring approach. The grammar of graphics further adds the plotting capacities to the calendar format.

1.2 Tidy temporal data structure

Wickham (2014) developed a set of tidy data principles to standardise and facilitate the data analysis process, and also suggested that each variable is a column, each observation is a row. An attribute describing a unit's properties (such as temperature, precipitation, pedestrian counts) is thought of as a variable. An observation refers to the same unit measured across each variable. A variable together with an observation defines the values that essentially comprise of a dataset. The newer concept of tidy data is the "long" form: every measured value is described by a unique set of identifiers. From the long form, data

can be summarised, and shaped, and arranged in many different forms, making different types of analysis on the same data easier.

However, in some ways it is the classical statistical format of a rectangular table where each time series is a column, and each time index is a row. This format is the "wide" form of tidy data, and it is what is typically required for many statistical models as it proves computationally efficient when doing matrix operations.

The common data structure for time series data in statistical software, like R, is actually short-hand. For example, a time series defined as a `ts` object in R is a vector of values, annotated with a starting time and frequency. Associated with the data object are methods that can be applied to plot, summarise and model. However, this approach is model-centric rather than data-centric. Data typically doesn't come in this form. Data typically has a lot more with it, and to get it in this form requires extracting a small part of all the data, and maybe, even massaging it into regular time measurements. Working from a tidy format this specialist form could be created with several wrapper functions, whilst maintaining the connections to the complete data.

1.3 Missingness in time

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 R package with functionality to organize and display temporal data, collected on sub-daily resolution, into a calendar layout. The function `frame_calendar` uses linear algebra on the date variable to restructure data into a format lending itself to calendar layouts. 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, was insufficient to examine the behavior. Making displays on a monthly calendar format helps to understand pedestrian patterns relative to events such as work days, weekends, holidays, and special events. The layout algorithm has several format options and variations. It is implemented in the R package **sugrrants**.

2.1 Introduction

We develop a method for organizing and visualizing temporal data, collected at sub-daily intervals, into a calendar layout. The calendar format is created using linear algebra,

giving a restructuring of the data, that can then be integrated into a data pipeline. The core component of the pipeline is to visualise the resulting data using the grammar of graphics (Wilkinson, 2005; Wickham, 2009), as used in **ggplot2** (Wickham et al., 2018a), which defines plots as a functional mapping from data points to graphical primitives. The data restructuring approach is consistent with the tidy data principles available in the **tidyverse** (Wickham, 2017) suite. The methods are implemented in a new R package called **sugrrants** (Wang, Cook, and Hyndman, 2018b).

The purpose of the calendar-based visualization is to provide insights into people's daily schedules relative to events such as work days, weekends, holidays, and special events. This work was originally motivated by studying foot traffic in the city of Melbourne, Australia (City of Melbourne, 2017). There have been 43 sensors installed that count pedestrians every hour across the inner-city area until the end of 2016 (Figure 2.1). The data set can shed light on people's daily rhythms, and assist the city administration and local businesses with event planning and operational management. A routine examination of the data would involve constructing conventional time series plots to catch a glimpse of temporal patterns. The faceted plots in Figure 2.2 give an overall picture of the foot traffic at three different sensors over 2016. Further faceting by day of the week (Figure 2.3) provides a better glimpse of the daily and sub-daily pedestrian patterns.

However, the conventional displays of time series data conceal patterns relative to special events (such as public holidays and recurring cultural/sport events), which may be worth noting to viewers.

The work is inspired by Wickham et al. (2012), which uses linear algebra to display spatio-temporal data as glyphs on maps. It is also related to recent work by Hafen (2018) 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 is some existing work in this area. For example, Van Wijk and Van Selow (1999) developed a calendar view of the heatmap to represent the number of employees in the work place over a year, where colors

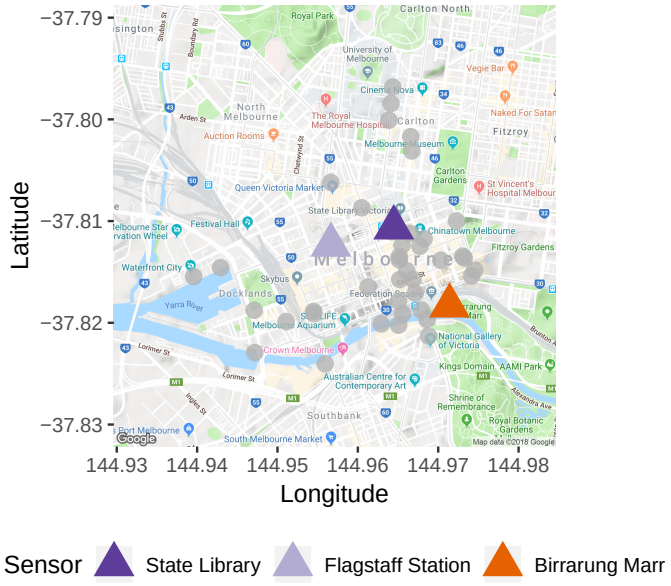

Figure 2.1: *Map of the Melbourne city area with dots indicating sensor locations. These three highlighted sensors will be inspected in the paper: (1) the State Library—a public library, (2) Flagstaff Station—a train station, closed on non-work days, (3) Birrarung Marr—an outdoor park hosting many cultural and sports events.*

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. Alongside Jones (2016), Wong (2013), Kothari and Ather (2016), and Jacobs (2017) implemented some variants of calendar-based heatmaps as in R packages: **TimeProjection**, **ggTimeSeries**, and **ggcal** 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. Additionally, if simply using colored blocks rather than curves, it may become perceptually difficult to estimate the shape positions and changes, although using curves comes with the cost of more display capacity (Cleveland and McGill, 1984; Lam, Munzner, and Kincaid, 2007).

We propose a new algorithm to go beyond the calendar-based heatmap. The approach is developed with three conditions in mind: (1) to display time-of-day variation in addition to longer temporal components such as day-of-week and day-of-year; (2) to incorporate line graphs and other types of glyphs into the graphical toolkit for the calendar layout; (3)

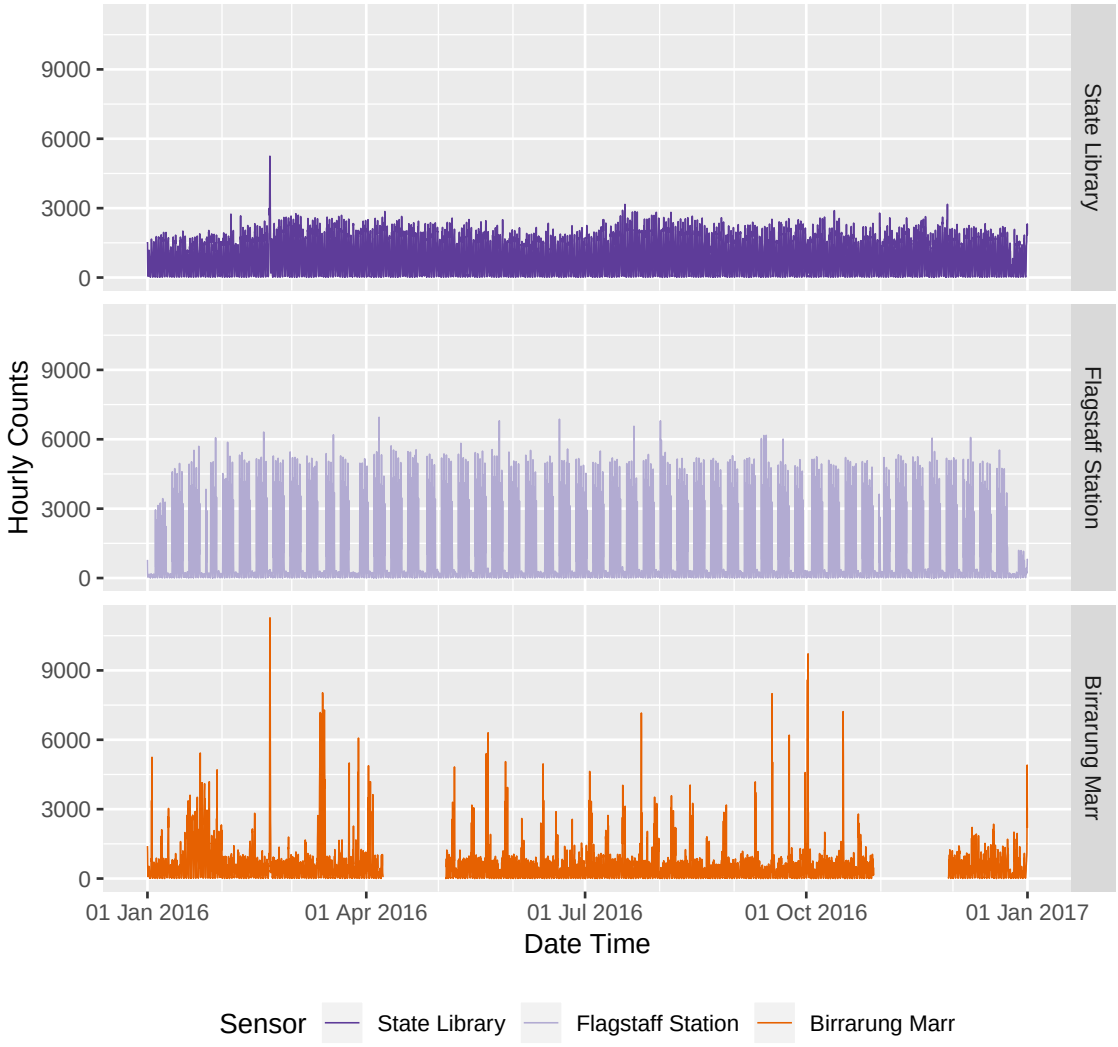

Figure 2.2: *Time series plots showing the number of pedestrians in 2016 measured at three different sensors in the city of Melbourne. Colored by the sensors, small multiples of lines show that the foot traffic varies from one sensor to another in terms of both time and number. A spike occurred at the State Library, caused by the annual White Night event on 20th of February. A relatively persistent pattern repeats from one week to another at Flagstaff Station. Birrarung Marr looks rather noisy and spiky, with a couple of chunks of missing records.*

to enable overlaying plots consisting of multiple time series. The proposed algorithm has been implemented in the `frame_calendar` function in the **sugrrants** package using R.

The remainder of the paper is organized as follows. Section 2.2 demonstrates the construction of the calendar layout in depth. Section 2.2.1 describes the algorithms of data transformation. Section 2.2.2 lists and describes the options that come with the `frame_calendar`

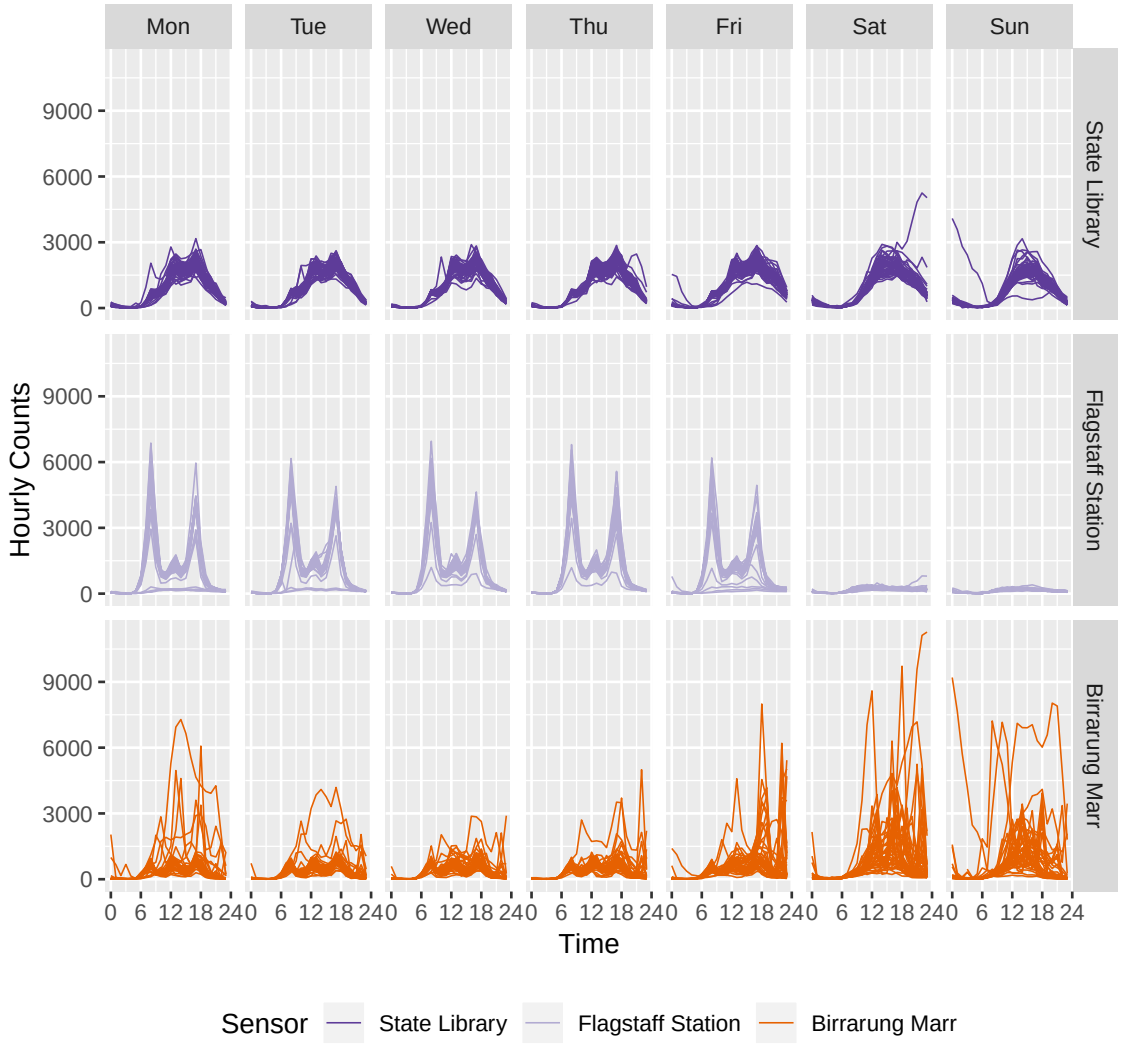

Figure 2.3: *Hourly pedestrian counts for 2016 faceted by sensors and days of the week using lines. It primarily features two types of seasons—time of day and day of week—across all the sensors. Apparently other factors have influence over the number of pedestrians, which cannot be captured by the faceted plots, such as the overnight White Night traffic on Saturday at the State Library and a variety of events at Birrarung Marr.*

function. Section 2.2.3 presents some variations of its usage. Graphical analyses of sub-daily people's activities are illustrated with 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 can be captured with 35 (5×7) cells, where the top left is Monday of week 1, and the bottom right is Sunday of week 5 by default. These cells provide a micro canvas on which to plot the data. The first day of the month could be any of Monday–Sunday, which is determined by the year of the calendar. Months are of different lengths, ranging from 28 to 31 days, and each month could extend over six weeks but the convention in these months is to wrap the last few days up to the top row of the block. The notation for creating these cells is as follows:

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

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 $k = 5$.

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 $N = 4$.

				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.*

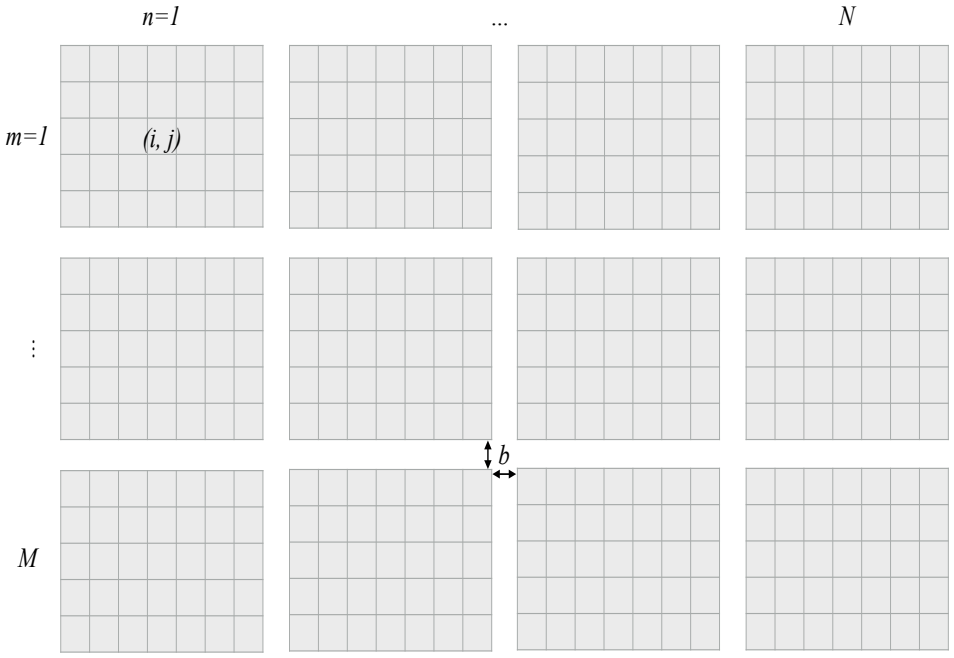

Figure 2.5: *Illustration of the indexing layout for months of one year, where M and N indicate number of rows and columns, b is a space parameter separating cells.*

Each cell forms a canvas on which to draw the data. Initialize the canvas to have limits $[0, 1]$ both horizontally and vertically. For the pedestrian sensor data, within each cell, hour is plotted horizontally and count is plotted vertically. Each variable is scaled to have

values in $[0, 1]$, using the minimum and maximum of all the data values to be displayed, assuming fixed scales. Let h be the scaled hour, and c the scaled count.

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

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

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

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., 2018a) functions. All of 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 $\min(y)$.

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.

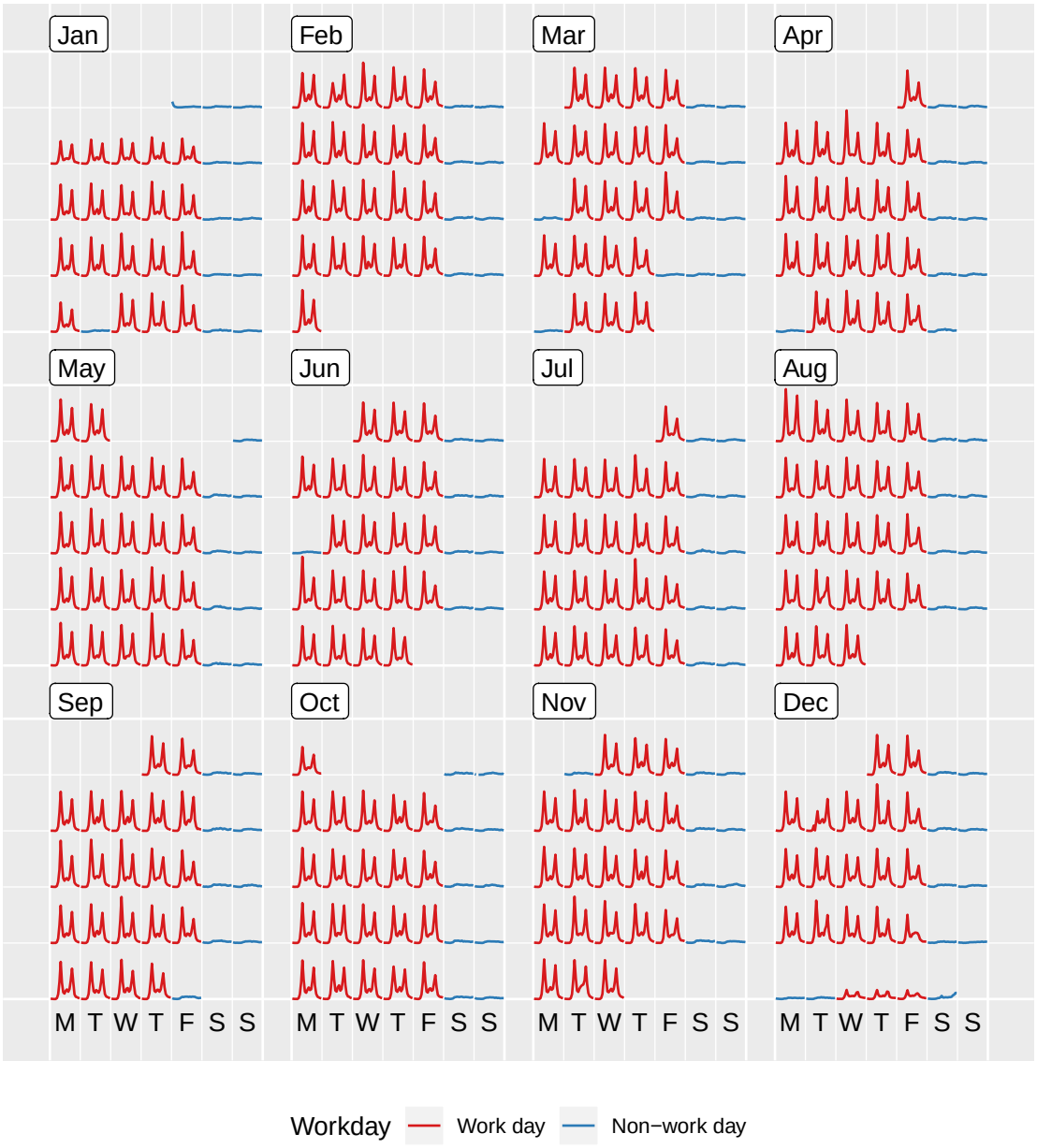

Figure 2.6: *The calendar-based display of hourly foot traffic at Flagstaff Station using line glyphs. The disparities between week day and weekend along with public holiday are immediately apparent. The arrangement of the data into a* 3×4 *monthly grid represents all the traffic in 2016. Note that the 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.*

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",
  sunday = FALSE, 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.

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). The resulting plot is star glyphs embedded in the monthly calendar layout. Star glyphs are time series lines transformed in polar coordinates.

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 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.

The `free_wday` scales each week day together. It can be useful to comparing 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.

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.

Language support

Most countries have adopted this western calendar layout, while the languages used for week day and month would be different across countries. We also offer language specifications other than English for text labelling.

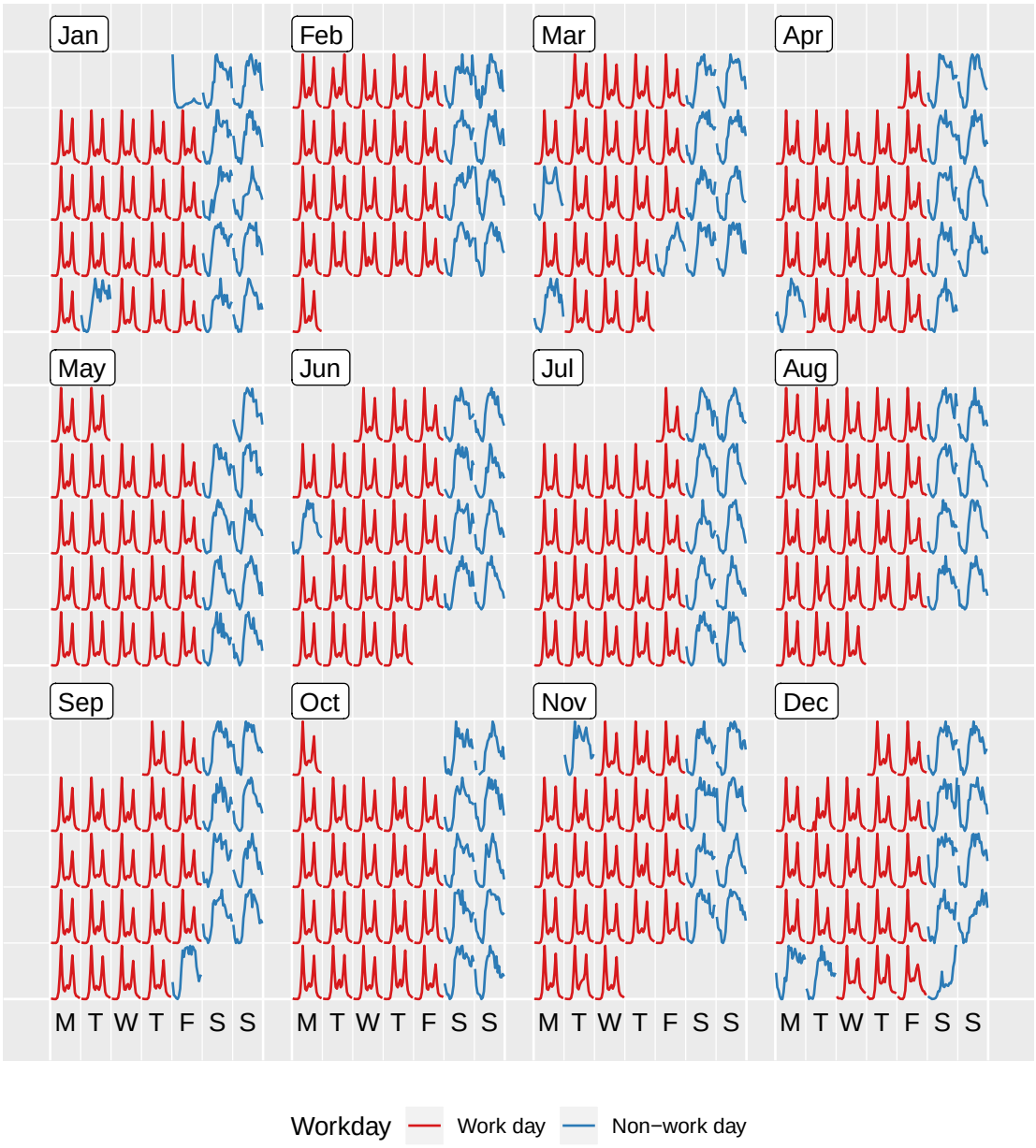

Figure 2.7: *Line glyphs on the calendar format showing hourly foot traffic at Flagstaff Station, scaled over all the days. The individual shape on a single day becomes more distinctive, however it is impossible to compare the size of peaks between days.*

2.2.3 Variations

Overlaying and faceting subsets

Plots can be layered. The comparison of sensors can be done by overlaying the values for each (Figure 2.8). Differences between the pedestrian patterns at these sensors can be seen. Flagstaff Station exhibits strong commuters patterns, with fewer pedestrian counts

Figure 2.8: *Overlaying line graphs of the three sensors in the monthly calendar. Three sensors demonstrate very different traffic patterns. Birrarung Marr tends to attract many pedestrians for special events held on weekends, contrasting to the bimodal commuting traffic at Flagstaff Station.*

during the weekends and public holidays. This suggests that Flagstaff Station has limited functionality on non-work days. From Figure 2.8 it can be seen that Birrarung Marr has a distinct temporal pattern from the other two all year round. The nighttime events, such as White Night, have barely affected the operation of Flagstaff Station but heavily affected the incoming and outgoing traffic to the State Library and Birrarung Marr.

To avoid the overlapping problem, the calendar layout can be embedded into a series of subplots for the different sensors. Figure 2.9 presents the idea of faceting calendar plots. This allows comparing the overall structure between sensors, while emphasizing individual sensor variation. In particular, it can be immediately learned that Birrarung Marr was busy and packed, for example during the Australian Open, a major international tennis tournament, in the last two weeks of January. This is concealed in the conventional graphics.

Different types of plots

The `frame_calendar` function is not constrained to line plots. The full range of plotting capabilities in **ggplot2** is essentially available. Figure 2.10 shows a lag scatterplot at Flagstaff Station, where the lagged hourly count is assigned to the `x` argument and the current hourly count to the `y` argument. This figure is organized in the daily calendar layout. Figure 2.10 indicates two primary patterns, strong autocorrelation on weekends, and weaker autocorrelation on work days. At the higher counts, on week days, the next hour sees possibly substantial increase or decrease in counts, essentially revealing a bimodal distribution of consecutive counts, as supported by Figure 2.6.

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.

Interactivity

As a data restructuring tool, the interactivity of calendar-based displays can be easily enabled, as long as the interactive graphic 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 it in the calendar layout, 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

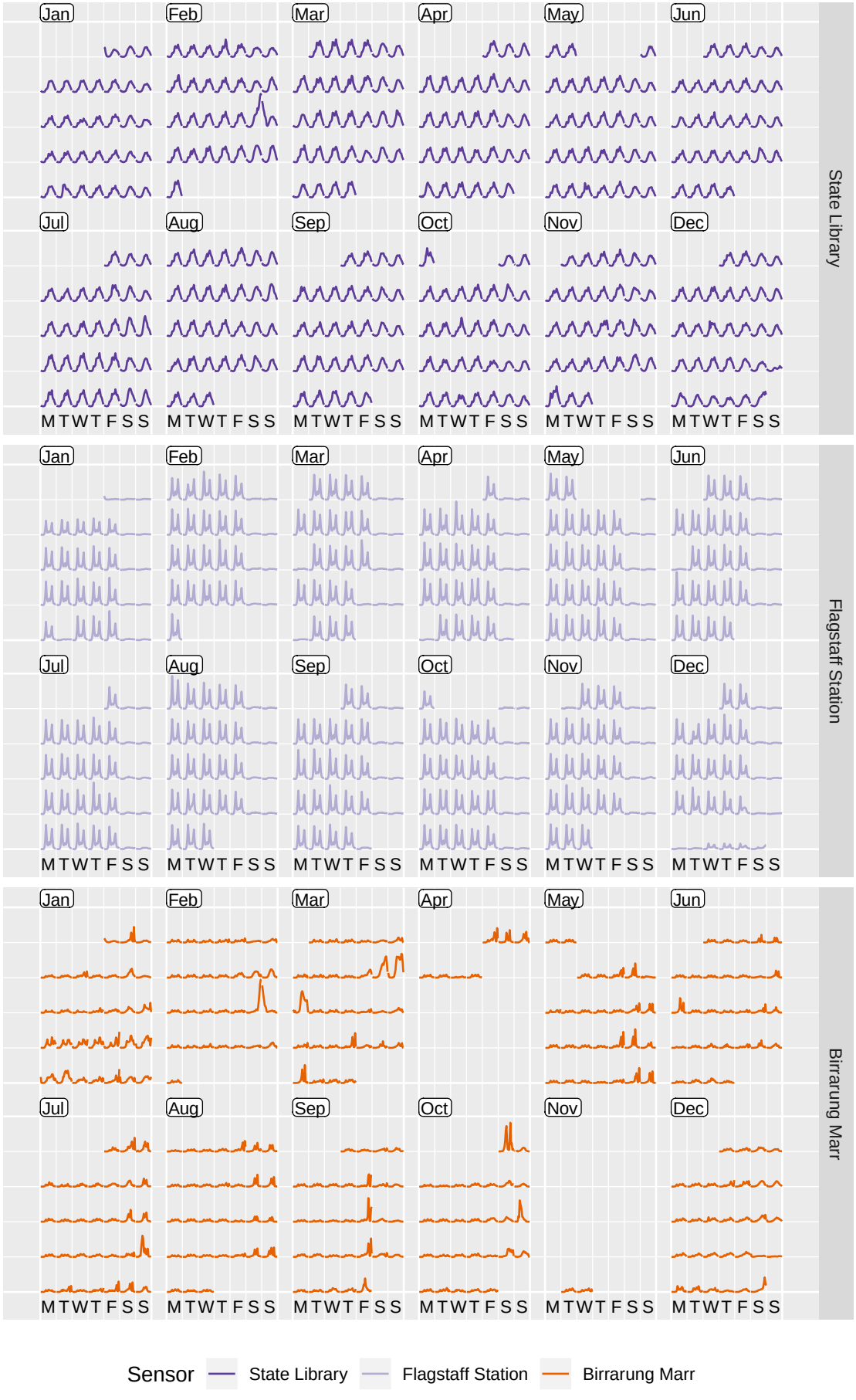

Figure 2.9: *Line charts, embedded in the 6 × 2 monthly calendar, colored and faceted by the 3 sensors. The variations of an individual sensor are emphasised, and the shapes can be compared across the cells and sensors.*

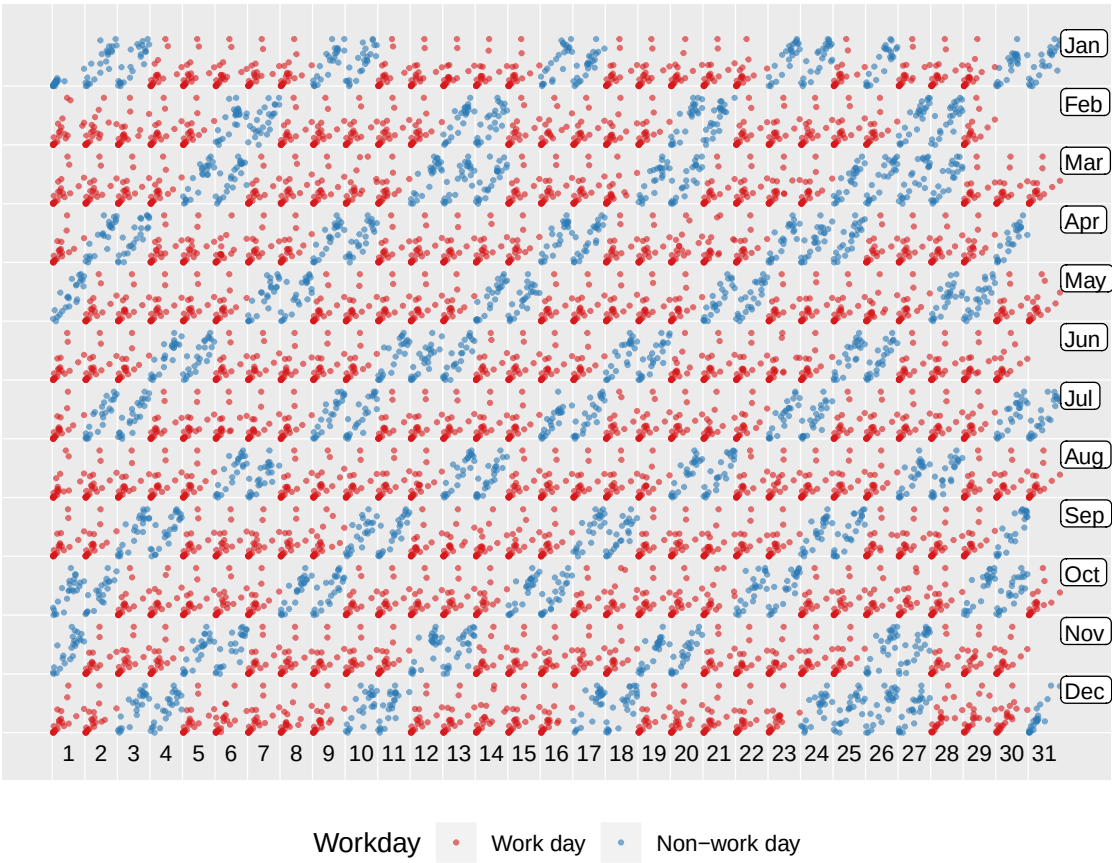

Figure 2.10: *Lag scatterplot in the daily calendar layout. Each hour's count is plotted against previous hour's count at Flagstaff Station to demonstrate the autocorrelation at lag 1. The correlation between them is more consistent on non-work days than work days.*

can be ported to a form of selection button or text input in a graphical user interface like R shiny (Chang et al., 2018). 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, 2018). 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 display is achieved using the common dates.

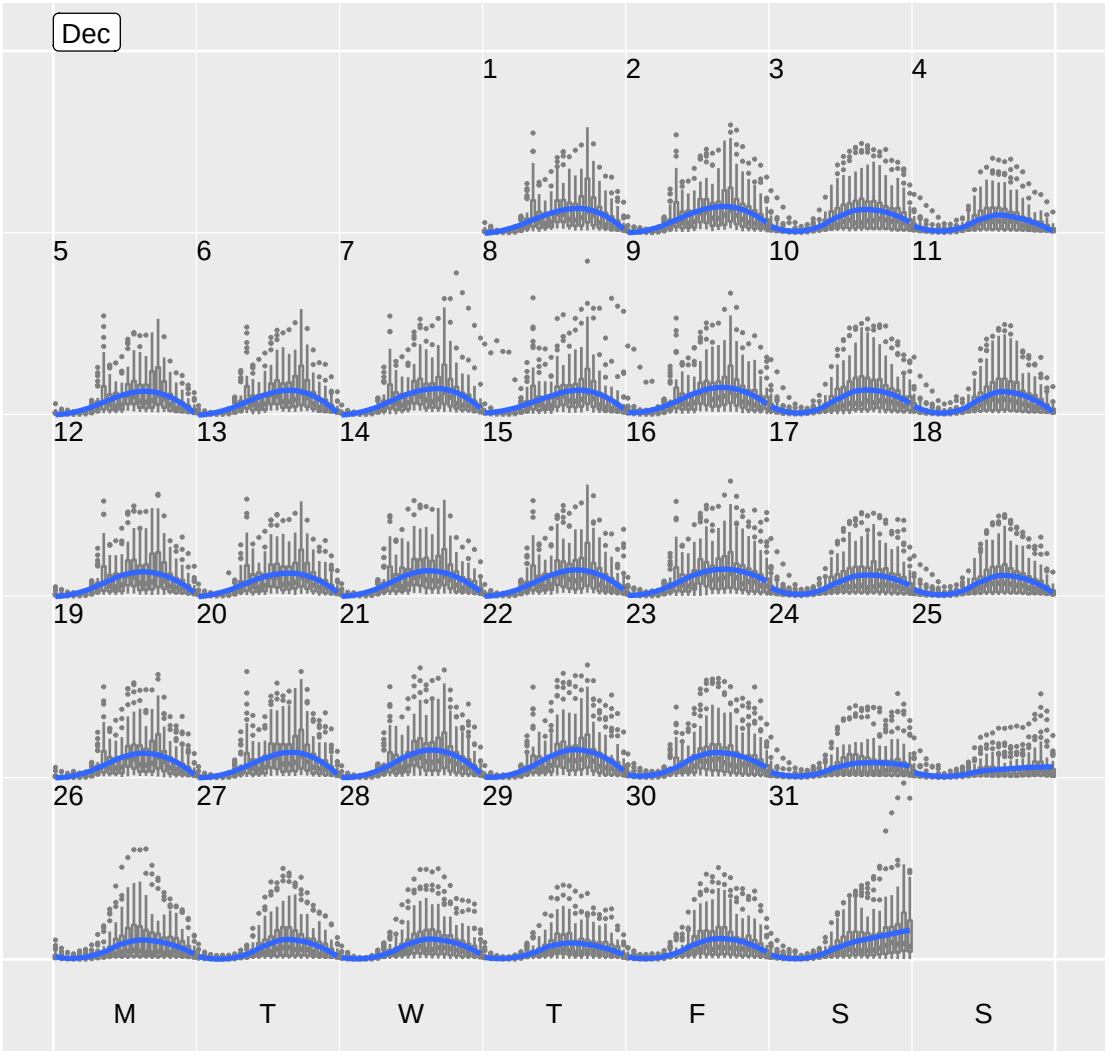

Figure 2.11: *Side-by-side boxplots of hourly counts for all the 43 sensors in December 2016, with the loess smooth line superimposed on each day. It shows the hourly distribution in the city as a whole. There is one sensor attracting a larger number of people on New Year's Eve than the rest.*

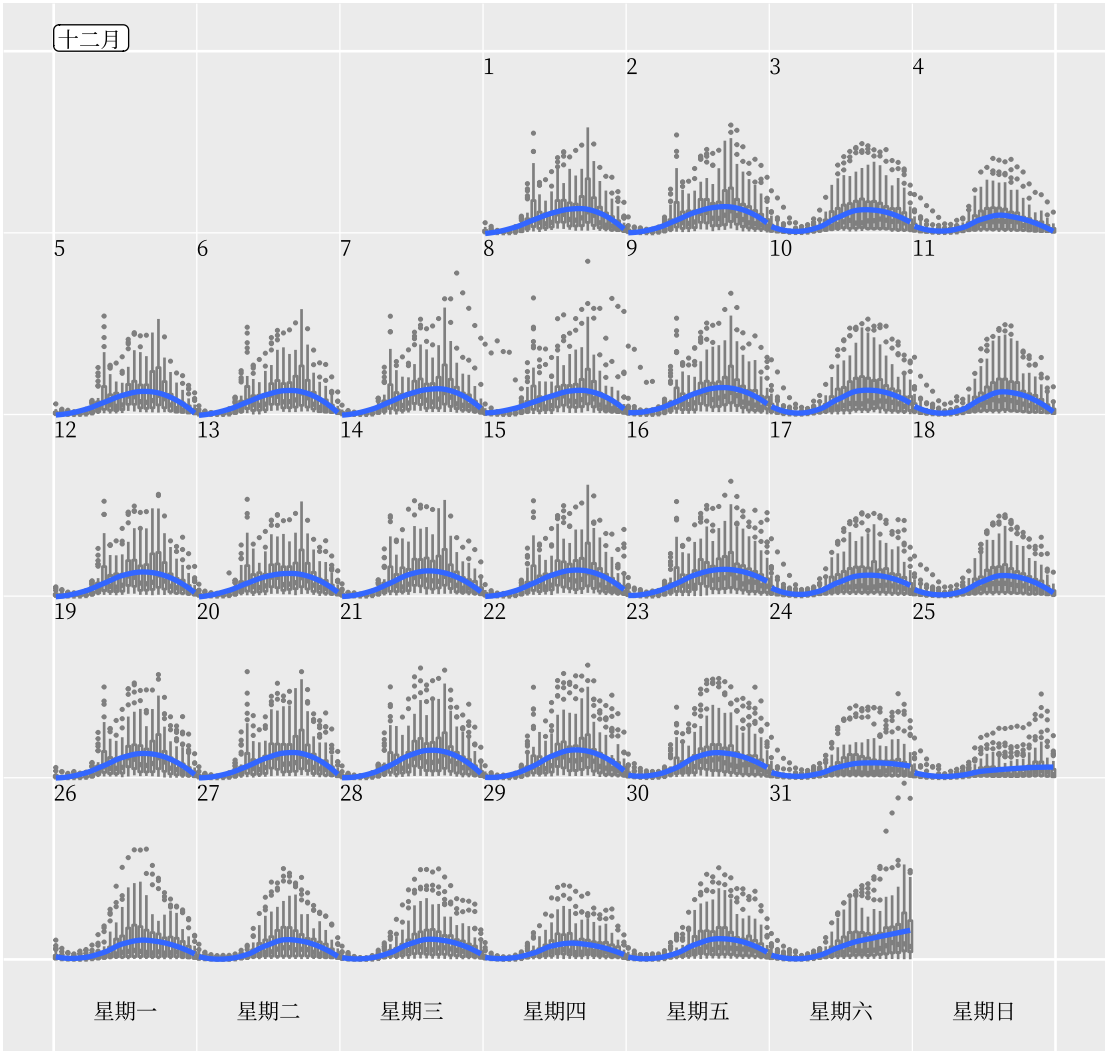

Figure 2.12: *The same plot as Figure 2.11, but with the month and week day labels in Chinese. It demonstrates the natural support for languages other than English.*

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 contained in the paper is made available 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 2017 and 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 in the form of letter value plots (Hofmann, Wickham, and Kafadar, 2017). Letter value plots are a variant of boxplots for large data, with other quantiles represented by boxes. Letters indicate the fraction of the data divisions, for example, F indicates fourths or quartiles, and the two outer ends of the box are the 25th and 75th percentile, the same traditional ends of the box as a boxplot. The letter E indicates eighths, with box ends being 12.5th and 87.5th percentiles of the data. These additional boxes replace the whiskers in a traditional boxplot. The letter value plots for the households, show a line indicating the median (M) and the innermost boxes corresponding to the fourth (F) and the eighth (E) quantile divisions. Inspecting the medians across households tells us that household 3, a family size of one couple and two kids, uses more energy over the week days, than other households. The relatively larger boxes for household 2 indicates 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 of 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 bird, starting their day before 6 and going back home around 18 on week days. They switch air conditioning or heating on when they get home from work and keep it operating until mid-night, learned from the small horizontal cluster of points around 0.8 kWh. On the other hand, the stripes above 1 kWh for household 2

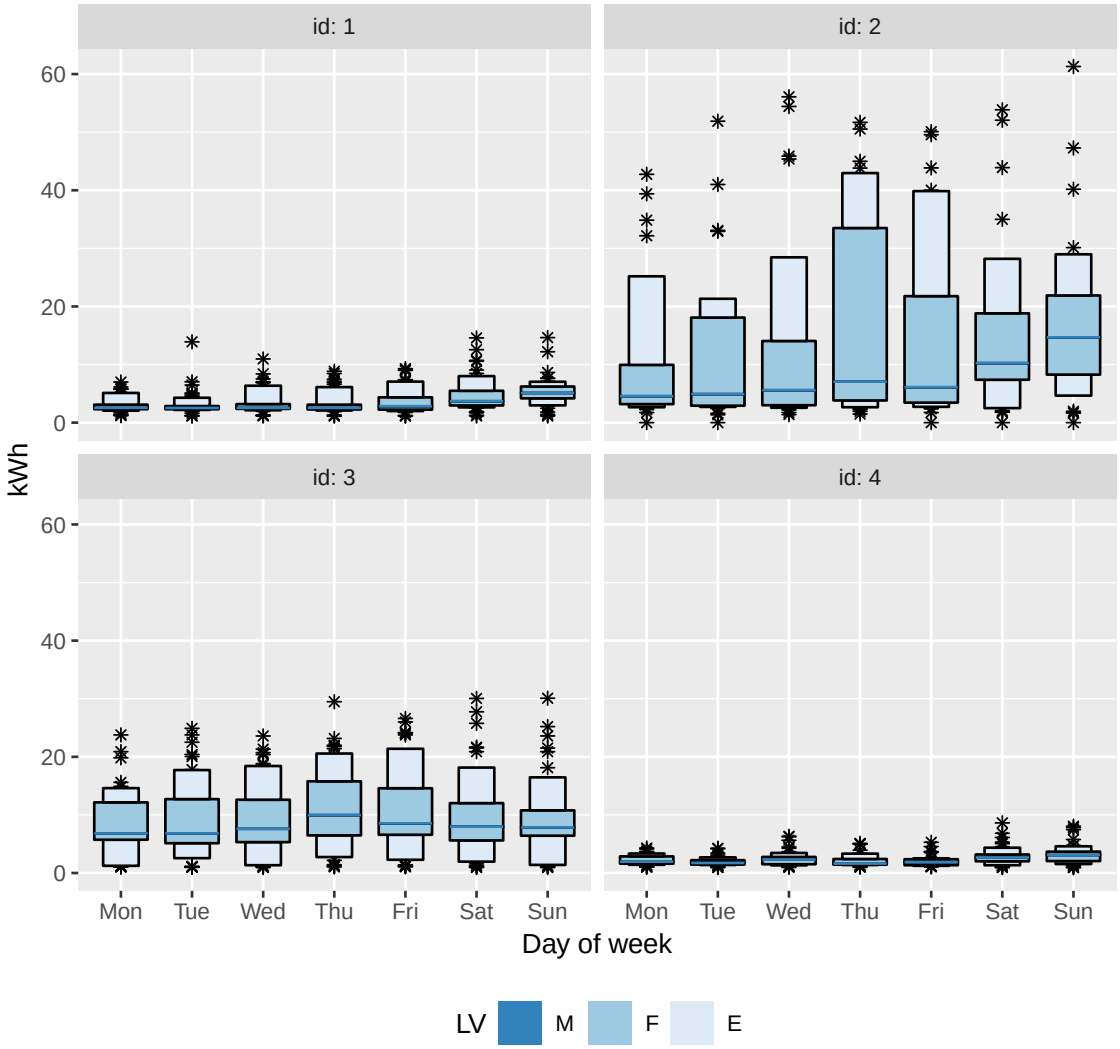

Figure 2.13: *Letter value plots of daily energy usage against day of week for four households, with one line dislaying the median (M) and each box corresponding to the fourth (F) and eighth (E) paired quantile estimates. Suggested by the medians, household 3 uses more enery than the others on the week days, due to a large family size. By constrast, household 2 sees considerally large variability.*

indicates that perhaps air conditioning or heating runs continuously for some periods, consuming the twice the energy as household 1. A third peak occurs around 15 for household 3 only, likely when the kids are home from school. They also have a consistent energy pattern between week days and weekends. As for household 4, their home routine starts after 18 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.

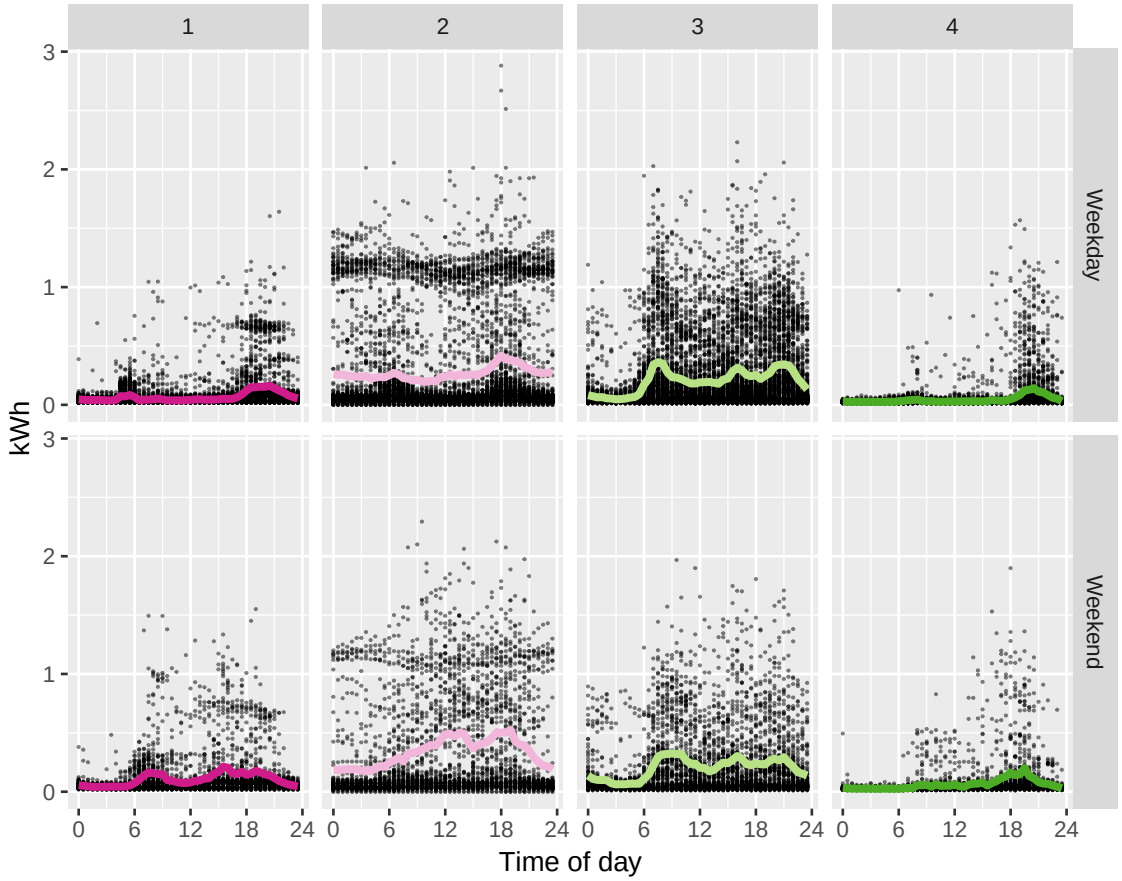

Figure 2.14: *Scatterplot between half-hourly energy usage and time of day with the averages overlaid, constrasting week days and weekends for each household. All have different daily routines between week days and weekends, except for household 3. On week days, household 1 wakes up early before 6, and household 2 around 6, followed by household 3 and 4. The use of air conditioning and heating are noted in household 1 and 2.*

Figure 2.15-2.18 display the data in calendar layout individually for each household, unfolding their day-to-day life. Glancing over household 1, we can see that their overall energy use is low. Their week day energy use is distinguishable from their weekends, indicating a working household. The air conditioner appears to be used in the summer months (December–February) in the evening and weekends. In household 2, heating keeps functioning for consecutive hours, which is evident in the mid July. In contrast, household 1 uses heating cautiously. These observations help to explain the stripes and clusters in Figure 2.14. The calendar plots speak the stories about vacation time that are untold by previous plots. Household 1 is on vacation over three weeks of June, and household 2 was also away for vacation in late December and in the second week of June.

Figure 2.15: *Calendar display for household 1, indicates higher weekend usage, and in the summer months, November-February. It seems that they took a vacation in June.*

Figure 2.17 shows household 3 takes two one-month-long family trips in September until early October and in June/July. Household 4 is away over two or three weeks in early October, December, early April, and late June. The use of air conditioning and heating leaves no trace in these two households.

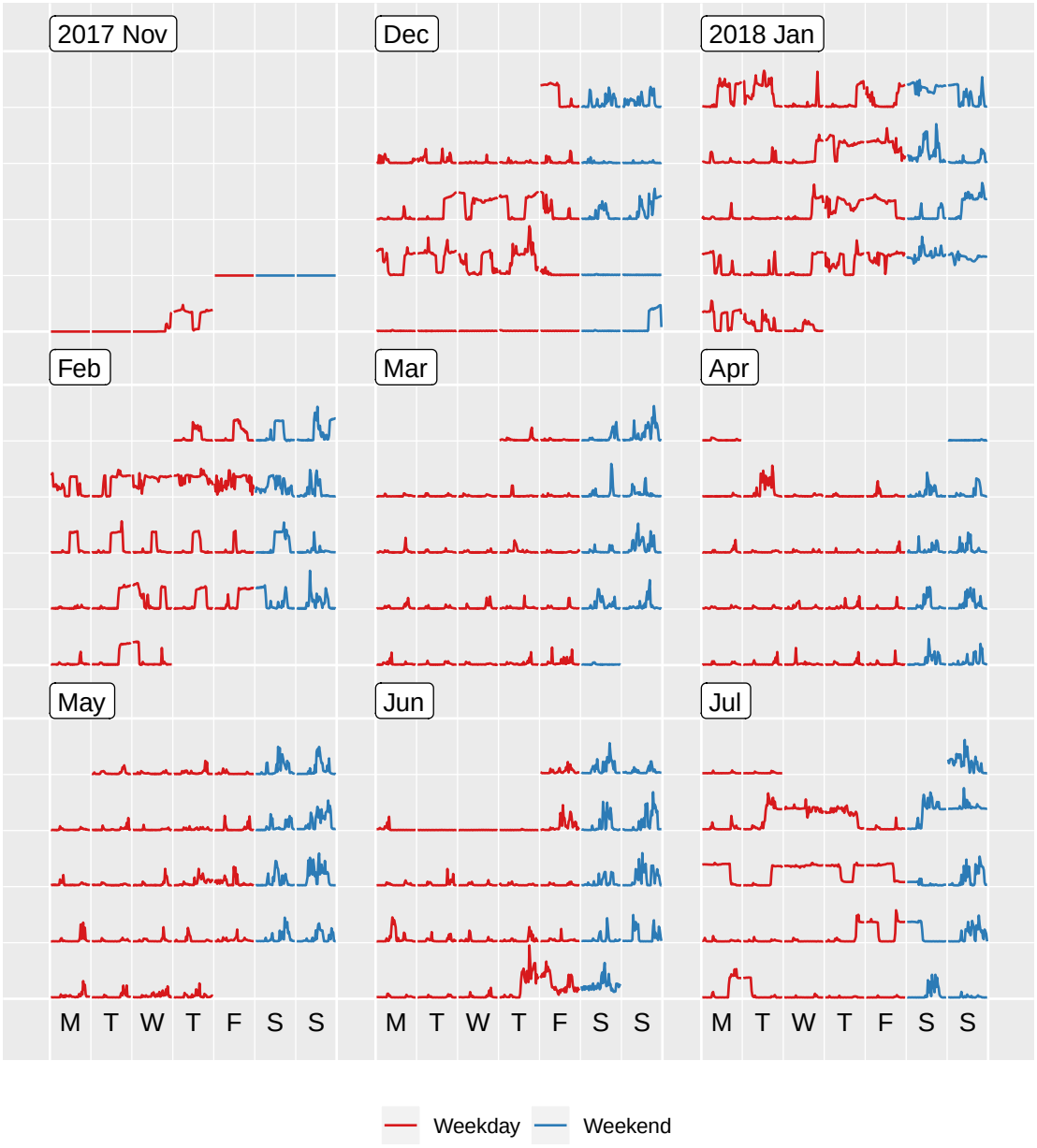

Figure 2.16: *Calendar display for household 2, reveals their tendency to use air conditioning and heating continuously. Not many vacation were taken.*

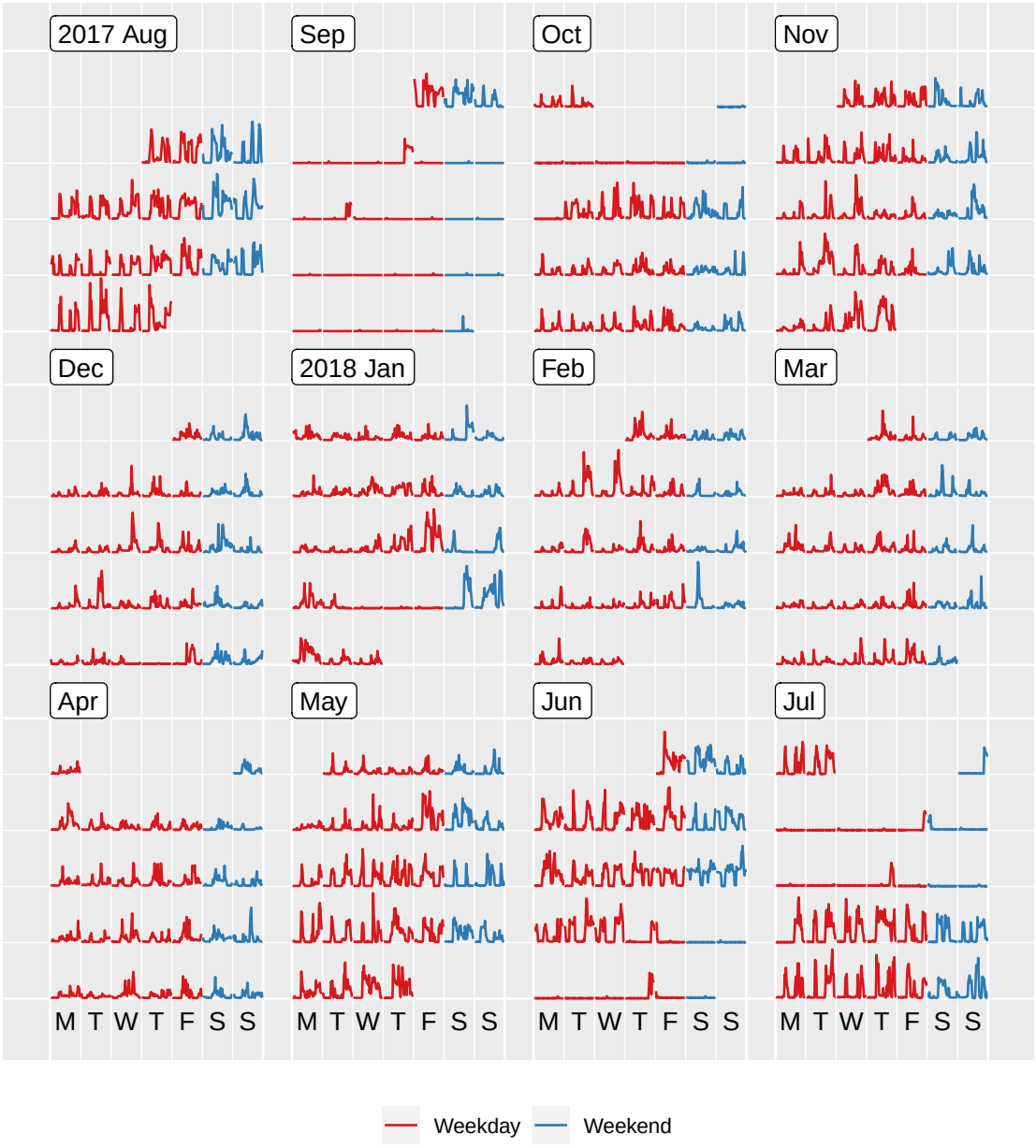

Figure 2.17: *Calendar display for household 3. Their energy use reveals higher energy use in the winter months, with multiple peaks daily on both week days and weekends. There are some high peaks in summer, perhaps indicating occasional air conditioner use. There have been several long vacations in the past year.*

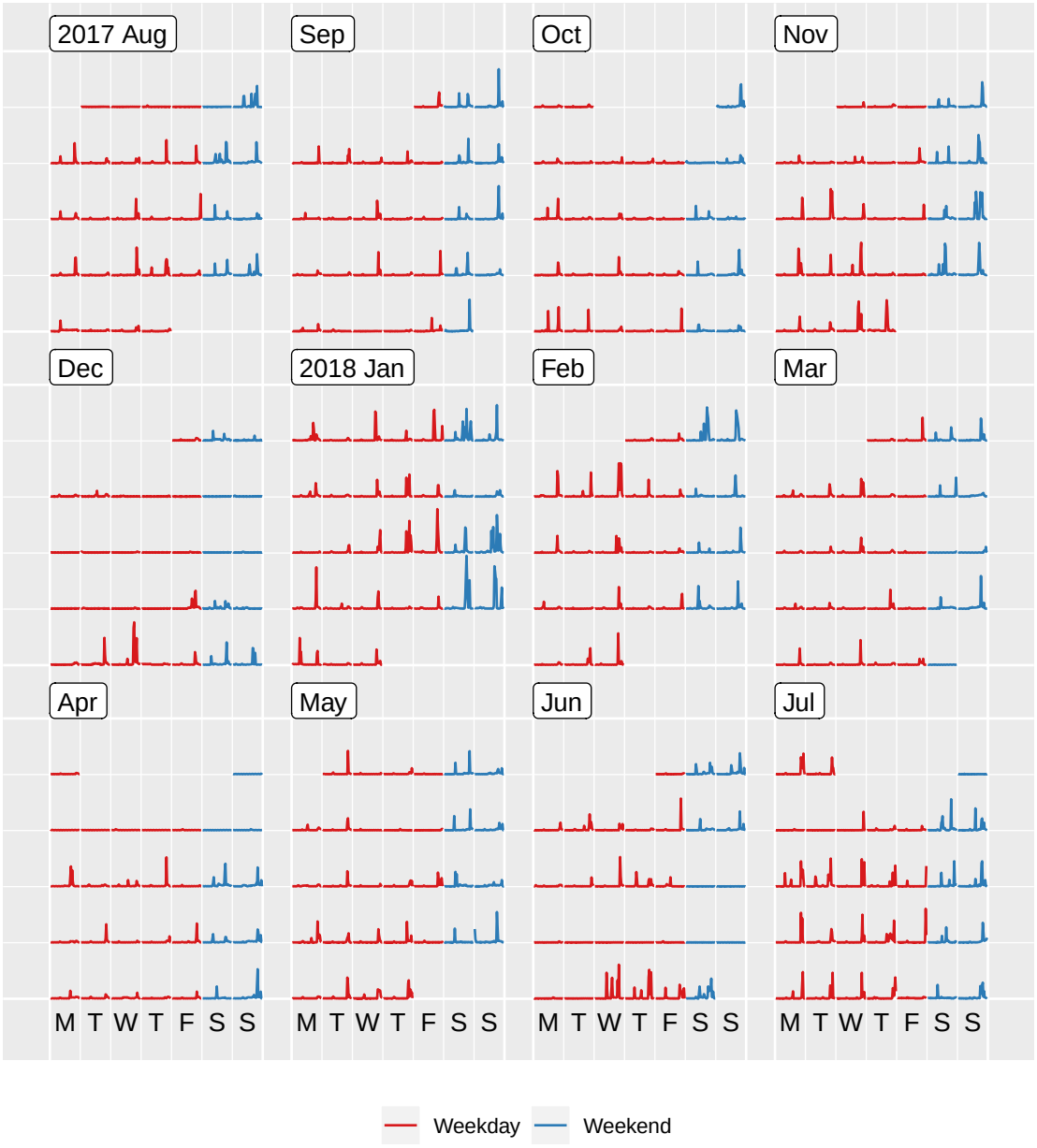

Figure 2.18: *Calendar display for household 4, shows energy use mostly in the evenings and on weekends. Three short trips were taken in October, December, April, and June.*

2.4 Discussion

The calendar-based visualization provides data plots in the familiar format of an everyday tool. Patterns on special events for the region, like Anzac Day in Australia, or Thanksgiving Day in the USA, are more visible to the viewer as public holidays.

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. Layouts beyond the western calendar could be achieved by the same modular arithmetic approach.

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

The layout is a type of faceting and could be useful to develop this into a fully-fledged faceting method with formal labels and axes. This is a future goal.

Acknowledgements

We would like to thank Stuart Lee and Heike Hofmann for their feedback about this work. The most recent version of the `frame_calendar` function is included in the **sugrrants** R package, which can be accessed via the CRAN website https://CRAN.R-project.org/package=sugrrants or 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: irregular or multiple time intervals, point events that need aggregating, multiple observational units or repeated measurements on multiple individuals, and heterogeneous data types. On the other hand, the software supporting time series modeling and forecasting, makes strict assumptions on the data to be provided, typically requiring a matrix of numeric data with implicit time indexes. Going from raw data to model-ready data is painful. 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 **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 summer 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 events, such as flight departures, that need to be reduced to a regular structure.

Despite this variety and heterogeneity of temporal data, current software typically requires time series objects to be model-oriented matrices. Analysts are expected to do their own data preprocessing and take care of anything else needed to allow model fitting, which leads to a myriad of ad hoc solutions and duplicated efforts.

Wickham and Grolemund (2016) proposed the tidy data workflow, which provides a conceptual framework for processing data (as described in Figure 3.1). Currently, time series modeling and forecasting enters this framework at the *modeling* stage, while temporal data enters at the start. This paper integrates time series analysis into this tidy framework, providing a coherent way for getting temporal data into the matrix format for modeling.

The paper is structured as follows. Section 3.2 reviews temporal data structures corresponding to time series and longitudinal analysis, and discusses "tidy data" and the grammar of data manipulation. Section 3.3 proposes contextual semantics for temporal data, built on top of tidy data. The concept of data pipelines with respect to the time domain will be discussed in depth in Section 3.4, followed by a discussion of the design choices made in the software structure in Section 3.5. Two case studies are presented in Section 3.6 illustrating temporal data exploration using the newly implemented infrastructure. Section 3.7 discusses future work.

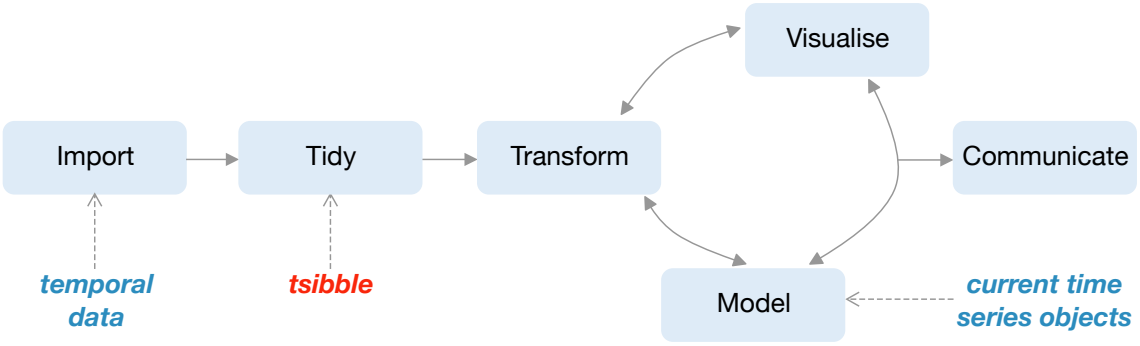

Figure 3.1: *Illustration of the data science workflow, drawn from Wickham and Grolemund (2016), showing how current time series tools interface with the workflow and how the tsibble structure and tools integrate. The new data structure, tsibble, makes the connection between temporal data input, and specialist modeling formats. It provides elements at the "tidy" step, which produce tidy temporal data for time series visualization and modeling.*

3.2 Data structures

3.2.1 Time series and longitudinal data

Temporal data problems are typically grouped into two types of analysis, time series and longitudinal. Despite being exactly the same data input, the representation of time series and longitudinal data diverges due to different modeling approaches.

Time series can be univariate or multivariate, and for modeling require relatively long lengths (i.e., large T). Time series researchers and analysts who are concerned with this large T property, are mostly concerned with stochastic processes, for the primary purpose of forecasting, and characterizing temporal dynamics. Most statistical software represent such time series as vectors or matrices. Multivariate time series are typically assumed to be in the format where each row is assumed to hold observations at a time point and each column to contain a single time series. (The tidy data name for this would be **wide format**.) This implies that data are columns of homogeneous types: numeric or non-numeric, but there are limited supporting methods for non-numeric variables. In addition, time indexes are stripped off the data and implicitly inferred as attributes or meta-information. There is a strict requirement that the number of observations must be the same across all the series. Data wrangling, from the form that data arrives in, to this specialist format, can be frustrating and difficult, inhibiting the performance of downstream tasks.

For longitudinal analysis, researchers and analysts are primarily interested in explaining trends across and variations among individuals, and making inference about a broader population. 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, resulting in an unbalanced panel. Thus, the primary format required for modeling such data is stacked series, blocks of measurements for each individual, with columns indicating individual, times of measurement and the measurements themselves. (The tidy data name for this would be **long format**.) Evidently, this data organization saves storage space for many sparse cells, compared to structuring it into wide format which would have missing values in many cells. A drawback of this format is that information unique to each individual is often repeated for all time points. An appealing feature is that data is structured in a semantic manner with reference to observations and variables, with the time index stated explicitly. This opens the door to easily operating on time to make calculations and extract different temporal components, such as month and day of the week. It is conducive to examining the data in many different ways and leading to more comprehensive exploration and forecasting.

3.2.2 Tidy data and the grammar of data manipulation

Wickham (2014) coined the term "tidy data", which is a rephrasing of the second and third normal forms in relational databases but in a way that makes more sense to data scientists by referring rows to observations and columns to variables. The principles of "tidy data" attempt to standardize the mapping of the semantics of a dataset to its physical representation. This 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., 2018b), 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, by composing them sequentially.

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). Temporal data tools need to catch up.

3.2.3 Existing time series standards in R

Current standards, provided by the native `ts` object in R, and extended by **zoo** (Zeileis and Grothendieck, 2005) and **xts** (Ryan and Ulrich, 2018), assemble temporal data into matrices with implicit time indexes. These objects were designed for modeling methods. The diagram in the style of Figure 3.1 would place the model at the center of the analytical universe, and all the transformations and visualizations would hinge on that format. This is contrary to the **tidyverse** conceptualization, which holistically captures the full data workflow.

A new temporal data class is needed in the upstream of the workflow, which could incorporate all the downstream modules. A relatively new R package **tibbletime** (Vaughan and Dancho, 2018b) proposed a data class of *time tibble* to represent temporal data in heterogeneous tabular format. It only requires an index variable to declare a temporal data object, thus placing it at the import stage. However, as proposed in Section 3.3 a more rigid data structure is required for time series analytics and models.

This paper describes a new tidy representation for temporal data, and a unified framework to streamline the workflow from data preprocessing to visualization and forecasting, as an integral part of a tidy data analysis.

3.3 Contextual semantics

The choice of tidy representation of temporal data arises from a data-centric perspective, which accommodates all of the operations that are to be performed on the data. Figure

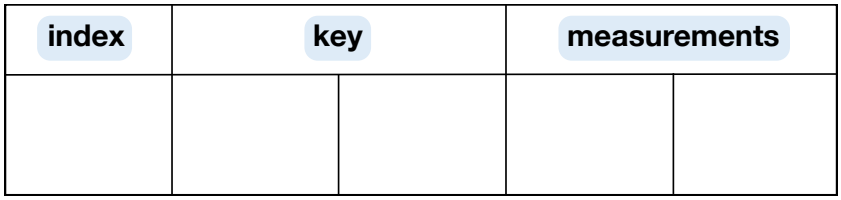

Figure 3.2: *The architecture of the tsibble structure is built on top of the data frame—a two-dimensional array in R, with time series semantics: index and key.*

3.1 marks where this new abstraction is placed in the tidy model, which we refer to as a "tsibble". The tsibble structure is an extension of a data frame—a two-dimensional array in R—with additional time series semantics: index and key, as shown in Figure 3.2.

To demonstrate the concept of the tsibble, Table 3.1 presents a subset of tuberculosis cases estimated by World Health Organization (2018). It contains 12 observations and 5 variables arranged in a "long" tabular 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. To turn this data into a tsibble: (1) column `year` is declared as the index variable; (2) the key is specified to consist of columns `country` and `gender`. The column `count` is the only measured variable in this data, but the structure is sufficiently flexible to hold other measured variables; for example, adding the corresponding population size (if known) in order to normalize the count later.

Table 3.1: *A small subset of estimates of tuberculosis burden generated 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 data structure, tsibble, bridges the gap between raw temporal data and model inputs. Contextual semantics are introduced to tidy data in order to support more intuitive time-related manipulations and enlighten new perspectives for time series model inputs. Index, key and time interval are the three stone pillars to this new semantically structured temporal data. Each is now described in more detail.

3.3.1 Index

Time provides a contextual basis for temporal data. A variable representing time is essential for a tsibble, and is referred to as an "index". The "index" is an intact data column rather than a masked attribute, which makes time visible and accessible to users. This is highly advantageous when manipulating time. For example, one could easily extract time components, such as time of day and day of week, from the index to visualize seasonal effects of response variables. One could also join other data sources to the tsibble based on common time indexes. The accessibility of the tsibble index motivates data analysis towards transparency and human readability. When the "index" is available only as meta information (such as in the `ts` class), it creates an obstacle for analysts to write these simple queries in a programmatic manner, which should be discouraged from an analytic point of view.

A variable number of time representations can be spotted in the wild. A date-time object, universally accepted across computing systems, is the most commonly used type for representing time. Date-time also typically associates with a time zone including adjustments such as summer time. This diversity and time zone is acknowledged and accommodated by tsibble's index. When creating a tsibble, time indices are arranged from past to future within each series for the strict temporal ordering that is assumed by time series operations.

3.3.2 Key

The "key" specification is the second essential ingredient for a tsibble. The "key" uniquely identifies observations that are recorded over time in a data table. It is similar to a primary key (Codd, 1970) defining each observation in a relational database. In the wide format in

which multiple time series are often structured, the columns hold a series of values, so that the column implicitly serves as identification. In long format, columns are melted with names converted to "key" values. However, the "key" provides much more flexibility. It is not constrained to a single field, but can be composed from multiple fields. The identifying variables from which the "key" is constituted remain the same as in the original table with no further tweaks.

The "key" is usually known a priori by analysts. For example, Table 3.1 describes the number of tuberculosis cases for each gender across the countries every year. This data description suggests that columns `gender` and `country` have to be declared as the key, similar to a panel variable for longitudinal data. Lacking either of the two will be inadequate, because the observations would not be uniquely identified, and thus a tsibble construction would fail. An alternative specification of the key for this data is to include a third variable `continent`. Since `country` is nested within `continent`, it is a free variable for use. This variable brings additional information that can be used for forecasting reconciliation (Hyndman and Athanasopoulos, 2017). The key needs to be explicit when multiple units exist in the data. The key can be implicit when it finds a univariate series in the table, but it cannot be absent from a tsibble.

The "key" also provides a link between the data, models, and forecasts. This neatly decouples the data from models and forecasts, leaving more room for necessary model components, such as coefficients, fitted values and residuals. More details are given in Section 3.4.3.

3.3.3 Interval

One of the cornerstones of time series data, and hence beneath a tsibble, is the time interval. This information plays a critical role in computing statistics (e.g. seasonal unit root tests) and building models (e.g. seasonal ARIMA). The principal divide is between regularly or irregularly spaced observations in time. A tsibble permits implicit missing time, making it difficult to distinguish regularity from the index. It relies on a user's specification by switching the `regular` argument off, when the data involves irregular intervals. This

type of data can flow into event-based data modeling, but would need to be processed or regularized to fit models that expect time series.

For data indexed in regular time space, the time interval is automatically calculated, by first computing absolute differences of time indexes and then finding the greatest common divisor. This covers all conceivable cases, assuming that all observations in a tsibble have only one interval. Data collected at different intervals should be organized in separate tsibbles, encouraging well-tailored analysis and models, because each observation may have different underlying data generating processes.

3.4 Data pipelines

A data pipeline describes the flow of data through an analysis, and can generally assist in conceptualizing the process, when it is applied to a variety 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 on the basis of their standard streams, so that the output of each program becomes the input to another. The Extract, Transform, and Load (ETL) process from recent data warehousing literature dating back to 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 **cranvas**, respectively. The pipeline is typically described with a one way flow, from data to plot. For interactive graphics, where all plots need to be updated when a user interacts with one plot, the events typically trigger the data pipeline to be run. Xie, Hofmann, and Cheng (2014) uses a reactive programming framework, to implement the pipeline, in which user's interactions trigger a sequence of modules to update their views, that is, practically the same as running the data pipeline producing each plot.

Building a data pipeline is technically difficult: many implementation decisions have to be made about the interface, input and output objects and functionality. The tidy data abstraction lays the plumbing for data analysis modules of transformation, visualization and modeling. Each module communicates with the others, requiring tidy input, producing tidy output, chaining a series of operations together to accomplish the analytic tasks.

What is notable about an effective implementation of a data pipeline is that it coordinates a user's analysis making it cleaner to follow, and permits a wider audience to focus on the data analysis without getting lost in a jungle of computational intricacies. A fluent pipeline glues tidy data and the grammar of data manipulation together. It helps (1) break up a big problem to into manageable blocks, (2) generate human readable analysis workflow, (3) avoid introducing mistakes, at least making it possible to trace them through the pipeline. New data tools developed in the R package **tsibble** (Wang, Cook, and Hyndman, 2019) articulate the time series data pipeline, which shepherds raw temporal data through to time series analysis, and plots. More detailed explanations are given in the following sections, and the examples.

3.4.1 Time series transformation

Figure 3.3 illustrates the distinction of a time series pipeline from a regular data pipeline. It is highly recommended to check for identical entries of key and index before constructing a tsibble. Duplicates signal the data quality issue, which would likely affect subsequent analyses and hence decision making. Analysts are encouraged to gaze at data early and reason about the process of data cleaning. When the data meets the tsibble standard, it flows neatly into the analysis stage and takes full advantage of the tsibble infrastructure.

Many time operations such as lag/lead and time series models, assume an intact vector input ordered in time. Since a tsibble permits time gaps in the index, it is good practice to check and inspect any gaps in time following the creation of a tsibble, in order to prevent inviting these avoidable errors into the analysis. The first suite of verbs (rephrasing actions performed on the object) are provided to understand and tackle implicit missing values: (1) `has_gaps()` checks if there exists time gaps; (2) `scan_gaps()` reveals all implicit missing observations; (3) `count_gaps()` summarizes the time ranges that are absent from the

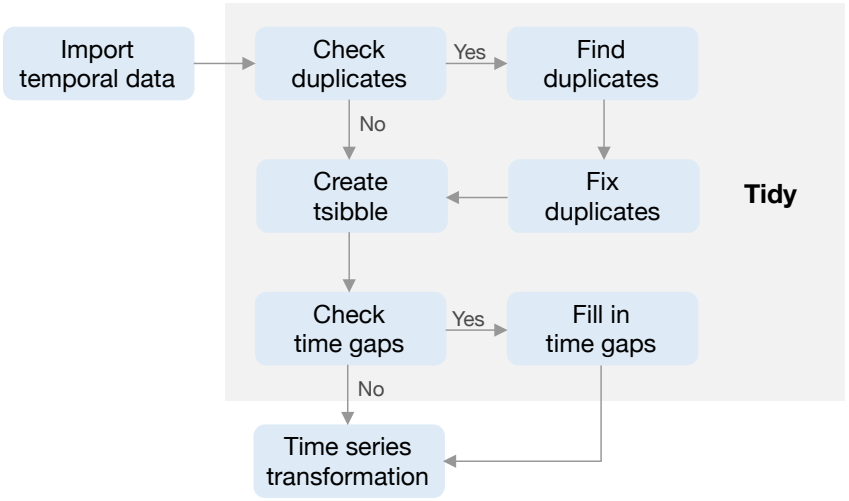

Figure 3.3: *A time series pipeline is different from a regular data pipeline: (1) check if the key and index uniquely identify each observation in the data whist creating a tsibble; (2) check if there are time gaps in the tsibble before analysis.*

data; (4) `fill_gaps()` turns them into explicit ones, along with imputing by values or functions. To look into gaps over individual time periods or full-length time span, the common argument `.full` in these functions gives an option to easily switch between. The specification of `.full = TRUE` will result in fully balanced panels in other words.

Besides the time gap verbs, the **tidyverse** vocabulary is adapted and expanded to facilitate time series transformations, as listed in Table 3.2. The **tidyverse** suite showcases the general-purpose verbs for effectively manipulating tabular data, for example `filter()` picks observations, `select()` picks variables, and `left_join()` joins two tables. But these verbs need handling with care when used in the time series domain. A perceivable difference is summarizing variables between data frame and tsibble using `summarize()`. The former will reduce to a single summary, whereas the latter will obtain the index and their corresponding summaries. Users who are already familiar with the **tidyverse**, will experience a gentle learning curve for mastering these verbs and glide into time series analysis with low cognitive load.

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 by automatically updating index and key under the hood. If the desired temporal ordering is changed by row-wise verbs, a warning is broadcast. If a tsibble cannot be maintained

Table 3.2: *A list of table verbs working with tsibble. Functions in bold originating from the tidyverse and are adapted for tsibble.*

	Verb	Description
Time gaps	`has_gaps()`	Test if a tsibble has gaps in time
	`scan_gaps()`	Reveal implicit missing entries
	`count_gaps()`	Summarize time gaps
	`fill_gaps()`	Fill in gaps by values and functions
Row-wise	**`filter()`**	Pick rows based on conditions
	`filter_index()`	Provide a shorthand for time subsetting
	`slice()`	Select rows based on row positions
	`arrange()`	Sort the ordering of row by variables
Column-wise	**`select()`**	Pick columns by variables
	`mutate()`	Add new variables
	`transmute()`	Drops existing variables
	`summarize()`	Aggregate values over time
Group-wise	`index_by()`	Group by index candidate
	`group_by()`	Group by one or more variables
	`group_by_key()`	Group by key variables
Reshape	**`gather()`**	Gather columns into long form
	`spread()`	Spread columns into wide form
	`nest()`	Nest values in a list-variable
	`unnest()`	Unnest a list-variable
Join tables	**`left_join()`** **`right_join()`** **`full_join()`** **`inner_join()`** **`semi_join()`** **`anti_join()`**	Join two tables together

in the output of a pipeline module (likely occurring to column-wise verbs), for example the index is removed by selection, an error informs users of the problem and suggests alternatives. This avoids surprising users and reminds them of the time context.

The tsibble structure and operations support data pipelines for sequencing analysis. Friedman and Wand (2008) asserted "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." Each verb works with other transformation family members in harmony. This set of verbs can result in many combinations to prepare tsibble for a broad range of visualization and modeling problems. Chaining operations is achieved with the pipe operator `%>%` introduced in the **magrittr** package (Bache and Wickham, 2014), read as "then". A sequence of functions can be composed in a way that can be naturally read from

left to right, which improves the readability of the code. It consequently generates a block of code without saving intermediate values.

Most importantly, a new ecosystem for tidy time series analysis has been undertaken, using the tsibble framework, and is called "tidyverts", a play on tidyverse that acknowledges the time series analysis purpose.

3.4.2 Time series visualization

As a tsibble is a subclassing of data frame, it integrates well with the grammar of graphics. It is easy to create and extend specialist time series plotting methods based on the tsibble structure, for example autocorrelation plots and calendar-based graphics (Wang, Cook, and Hyndman, 2018a).

3.4.3 Time series models

Modeling is crucial to explanatory and predictive analytics, but often imposes stricter assumptions on tsibble data. The verbs listed in Table 3.2 ease the transition to a tsibble that suits modeling. A tidy forecasting framework built on top of tsibble is under development, which aims at promoting transparent forecasting practices and concise model representation. A tsibble usually contains multiple time series. Batch forecasting will be enabled if a univariate model, such as ARIMA and Exponential Smoothing, is applied to each time series independently. This yields a "mable" (short for model table), where each model relates to each "key" value in tsibble. This avoids expensive data copying and reduces model storage. The mable is further supplied to forecasting methods, to produce a "fable" (short for forecasting table) in which each "key" along with its future time holds predictions. It also underlines the advantage of tsibble's "key" in acting as linkage between data inputs, models and forecasts. Advanced forecasting techniques, such as vector autocorrelation, hierarchical reconciliation, and ensembles, can be developed in a similar spirit. The modeling module is a current endeavor.

3.5 Software structure and design decisions

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 context, particularly useful when handling a large collection of data. The contextual summary provided by the print function, shown below on the data 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 series. These details 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:

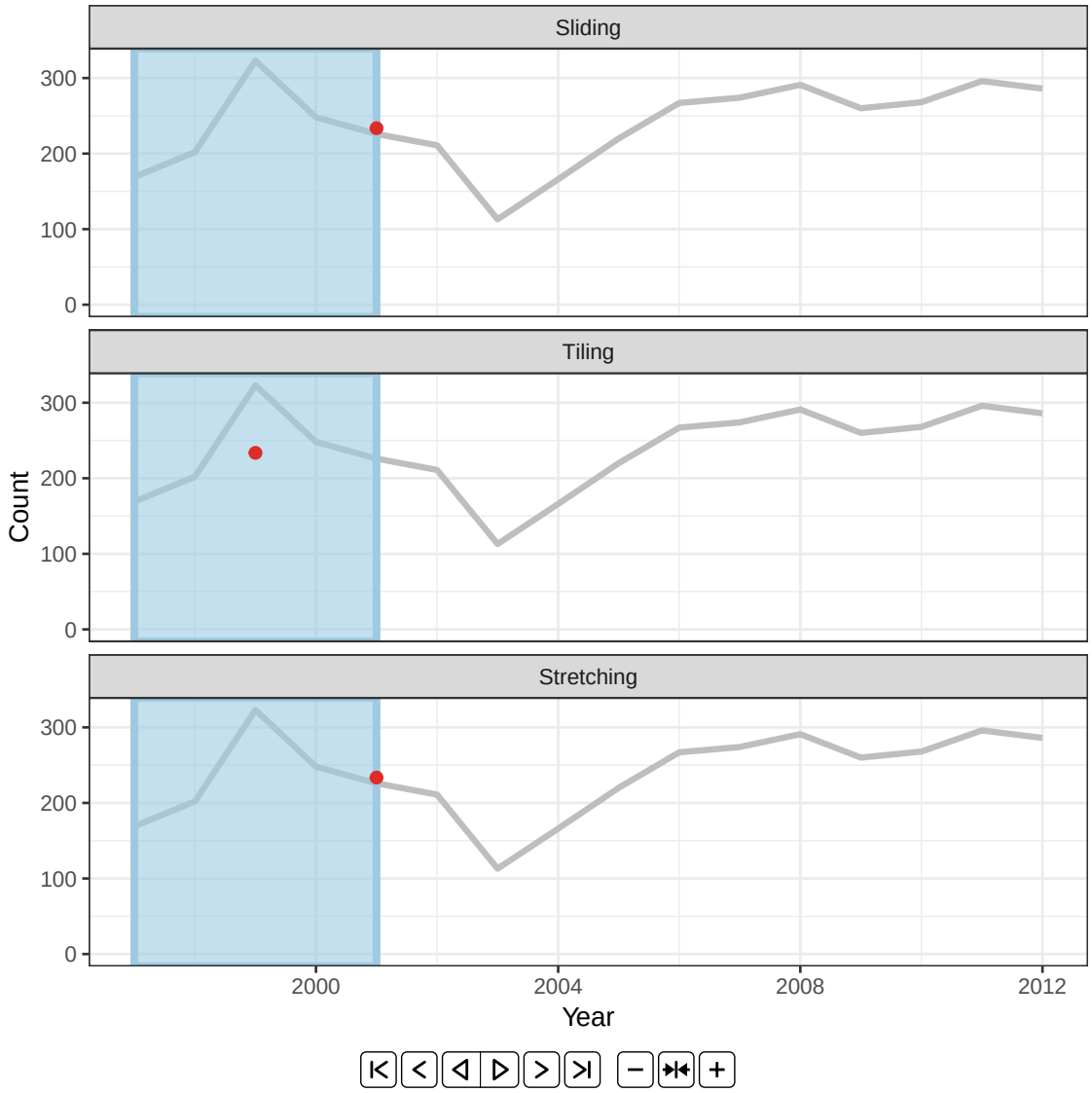

Figure 3.4: *An illustration of window of size 5 computing rolling averages over annual tuberculosis cases in Australia with respect to sliding, tiling and stretching. (Animation needs to be viewed with Adobe Acrobat Reader.)*

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 the animations of rolling windows for sliding, tiling and stretching, respectively, on annual tuberculosis cases for Australia. A block of consecutive elements with a window size of 5 are initialized and started rolling sequentially till the end of series by computing average counts.

Rolling window uses a programming paradigm—functional programming, which is different from those table verbs listed in Table 3.2. Table verbs expect and return a tsibble, and does what the function name suggests. On the contrary, these rolling window functions could accept arbitrary input types and would return arbitrary sorts of output, depending on which method is put into the rolling window. For example, computing moving averages requires numerics and a function like `mean()`, and produces averaged numerics. However, rolling window regression takes a data frame and a linear regression method like `lm()`, and generates a complex object that contains coefficients, fitted values, and etc.

The **purrr** package (Henry and Wickham, 2018) provides a good example of functional programming in R. It provides a complete and consistent set of tools to iterate each element of a vector with a function. Rolling window does not just iterate but rolls over a sequence of elements, namely `slide()`, `tile()` and `stretch()`. `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 type, 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). This family of functions empowers users to incorporate window-related operations in their workflows.

3.5.3 Modularity

Modular programming is adopted in the design of the **tsibble** package. Modularity benefits users by providing small focused and manageable 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 deal with time. The atomic functions (such as `yearmonth()` and `yearquarter()`) embedded in the `index_by()` table verb achieves in collapsing a tsibble to a less granular interval. The substitution of another time function in the `index_by()` results in the

aggregation of different time resolution. Since these time functions are not exclusive to a tsibble, they can be used in a variety of applications in conjunction with other packages. On the other hand, these tsibble verbs can incorporate many third-party vector functions to step out of the current tsibble zone. It is also generally easier to trace back the errors users encounter from separating 1d and 2d functions, and increase the code readability.

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, which heavily relies on S3 classes and methods in R (Wickham, 2018). The package can be extended in two major aspects: custom index and new tsibble class.

Time representation could be arbitrary, for example R's native `POSIXct` and `Date` for versatile date-times, nano time for nanosecond resolution implemented in **nanotime** (Eddelbuettel and Silvestri, 2018), and pure numbers in simulations. 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, within an academic year which is defined differently from a calendar year. A new index would be immediately recognized by the software upon defining `index_valid()`, as long as it can be ordered from past to future. The interval regarding semesters is further outlined through `pull_interval()`. As a result, the rest of the software 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 underlying basis of temporal data, and there is a demand for sub-classing a tsibble. For example, a fable is actually an extension of a tsibble, mentioned in Section 3.4.3. 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, like prediction distributions to a

fable. Tsibble verbs are also S3 generics. Developers will be able to implement these verbs for the new class, if necessary.

3.5.5 Tidy evaluation

The **tsibble** packages leverages the **tidyverse** grammars and pipelines through tidy evaluation (Henry and Wickham, 2019b) via the **rlang** package (Henry and Wickham, 2019a). 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 repeated references to the data source, but the resulting code is typically cleaner and more expressive.

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 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" via `id(flight_num)`. As a result of event timing, the data are 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 (`5,548,444 x 22 [!] <UTC>`) and the number of time-based 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-specific and general-purpose manipulation verbs can result in well-constructed pipelines. From the

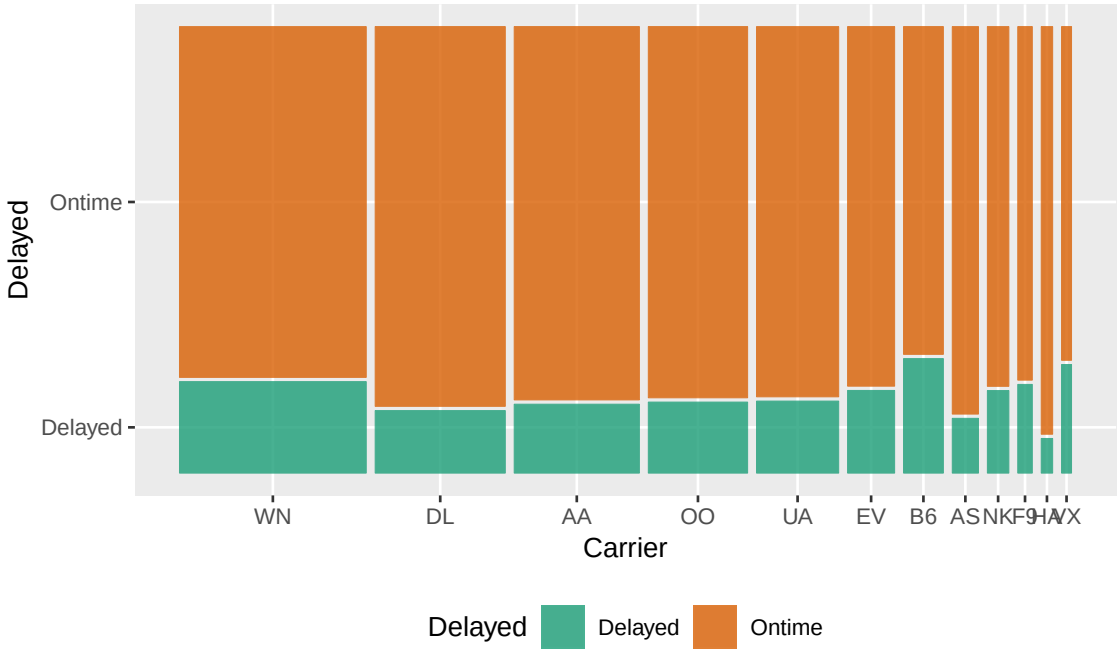

Figure 3.5: *Mosaic plot showing the association between the size of airline carriers and the delayed proportion of departures in 2017. Southwest Airlines is the largest operator, but does not operate as efficiently as Delta. Hawaiian Airlines, a small operator, outperforms the rest.*

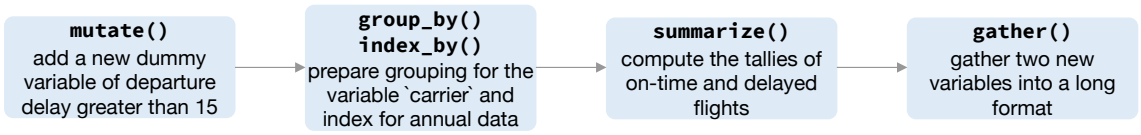

Figure 3.6: *Flow chart illustrating the pipeline that preprocessed the data for creating Figure 3.5.*

perspective of a passenger, for example, one needs to travel smart by choosing an efficient carrier to fly with and the time of day to avoid congestion. To explore this data, we drill down starting with annual carrier performance and followed by disaggregation to finer time resolutions.

Figure 3.5 visually presents the end product of aggregating the number of on-time and delayed flights to the year interval by carriers. This pipeline is initialized by defining a new variable if the flight is delayed, and involves summarizing the tallies of on-time and delayed flights for each carrier annually. To prepare the summarized data for a mosaic plot, it is further manipulated by melting new tallies into a single column. 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, and the output ends up with a tsibble of

unknown interval (as each carrier ends up with only one annual summary). The final data set includes each carrier along with a single year, with the interval undetermined, which in turn feeds into the mosaic display. Note that Southwest Airlines (WN), as the largest carrier, operates less efficiently than Delta (DL).

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 framed 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 visible. The number of days for each month is variable. Moving averages for two months call for computing weighted mean. But this can also be accomplished using a pair of commonly used verbs–`nest()` and `unnest()` to handle list-columns, without weight specification. The sliding operation with a large window size smooths out the fluctuations and gives a stable trend around 25% over the year. LAX airport has seen a gradual decline in delays over the year, whereas the SEA airport has a steady number delays over time. 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.

What time of day and day of week should we travel to avoid suffering from horrible delay? Figure 3.9 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 shown. To reduce the likelihood of suffering a delay, it is recommended to avoid the peak hour around 6pm (18).

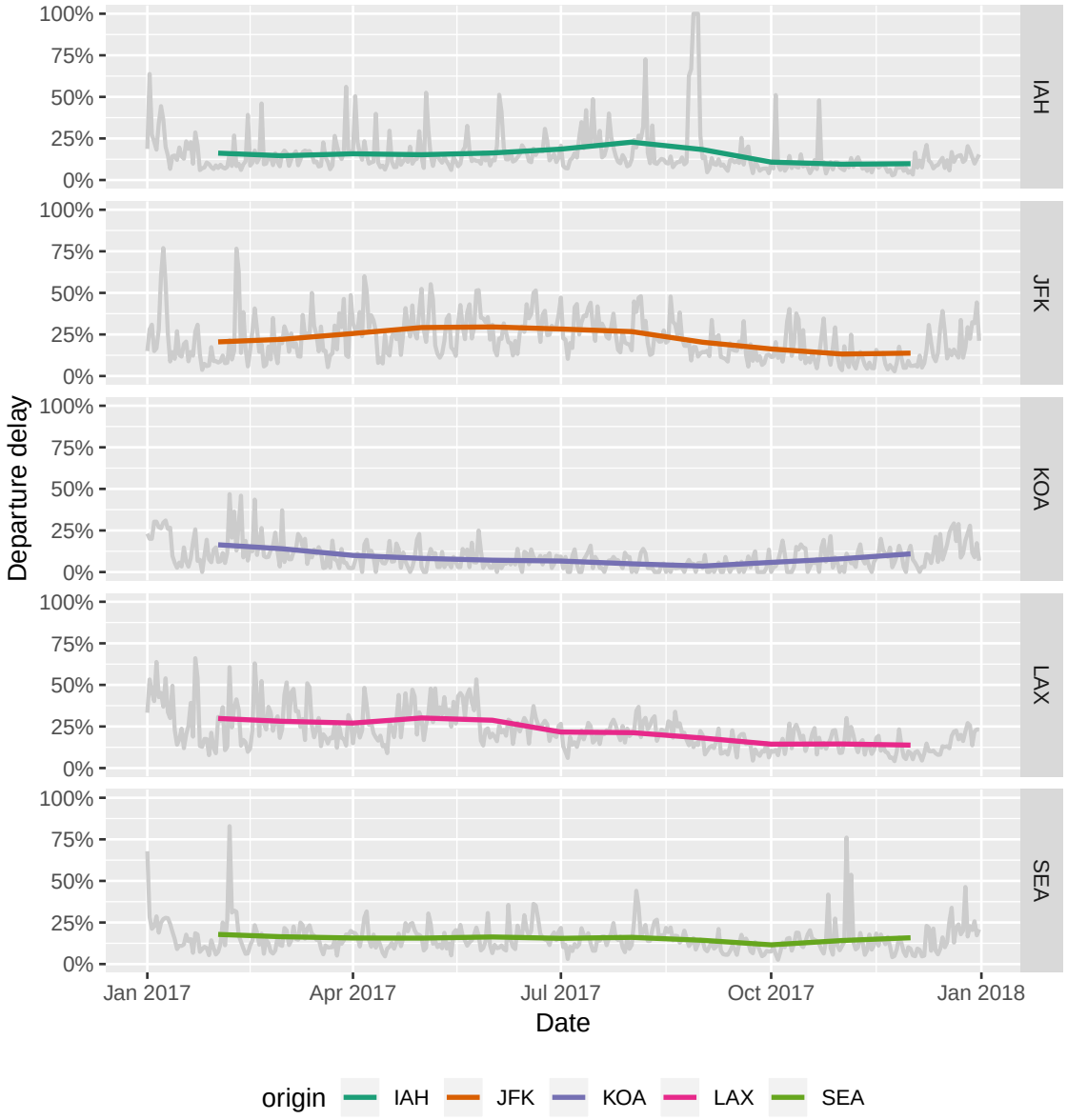

Figure 3.7: *Daily delayed percentages for departure with two-month moving averages overlaid at five international airports. There are least fluctuations and relatively fewer delays observed at KOA airport. The estimates of temporal trend are around 25% across the other four airports, but highlight different time periods of severe delays.*

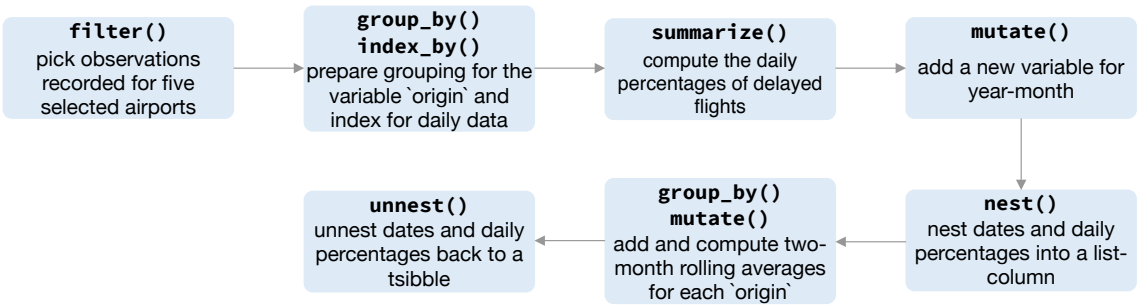

Figure 3.8: *Flow chart illustrating the pipeline that preprocessed the data for creating Figure 3.7.*

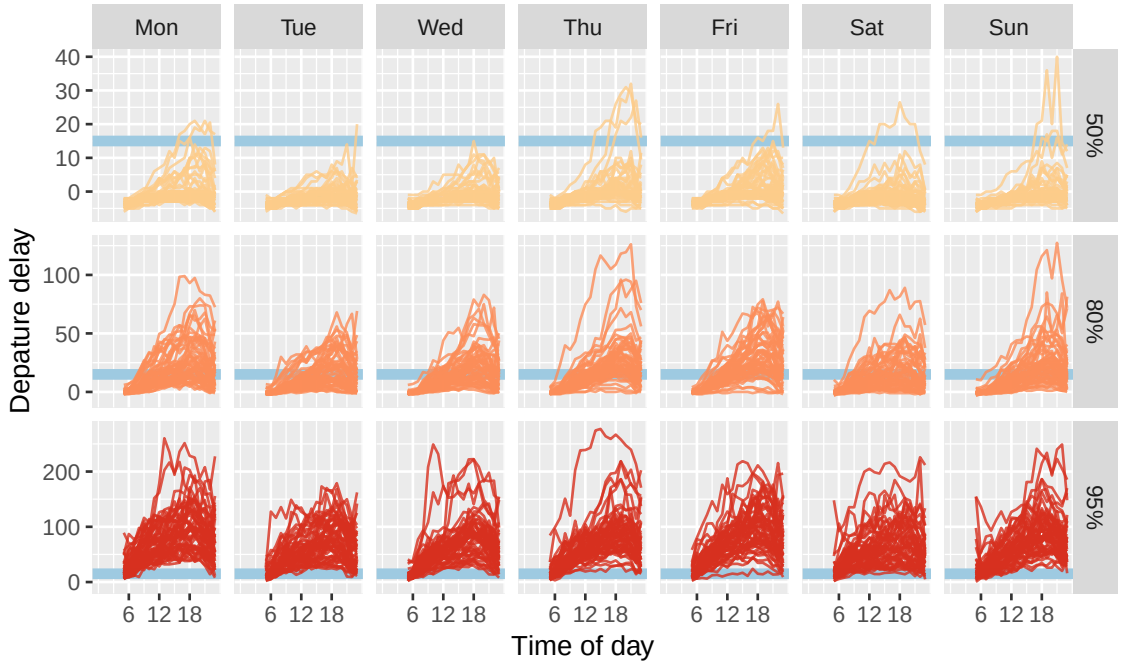

Figure 3.9: *Small multiples of lines about departure delay against time of day, faceting day of week and 50%, 80% and 95% quantiles. A blue horizontal line indicates the 15-minute on-time standard to help grasp the delay severity. Passengers are apt to hold up around 18 during a day, and are recommended to travel early. The variations increase substantially as the upper tails.*

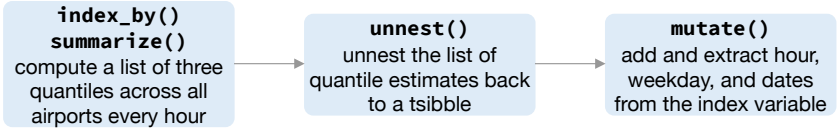

Figure 3.10: *Flow chart illustrates the pipeline that preprocesses the data for creating Figure 3.9.*

3.6.2 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 customer's 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 provides explanatory variables other than time in a different data table (Bureau of Meteorology, 2019), obtained using the R package **bomrang** (Sparks et al., 2018). Two data tables might be joined to explore how local weather can contribute to the variations of daily electricity use when needed.

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 3.11 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, organized into a calendar display (Figure 3.12). 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 predominant driven by

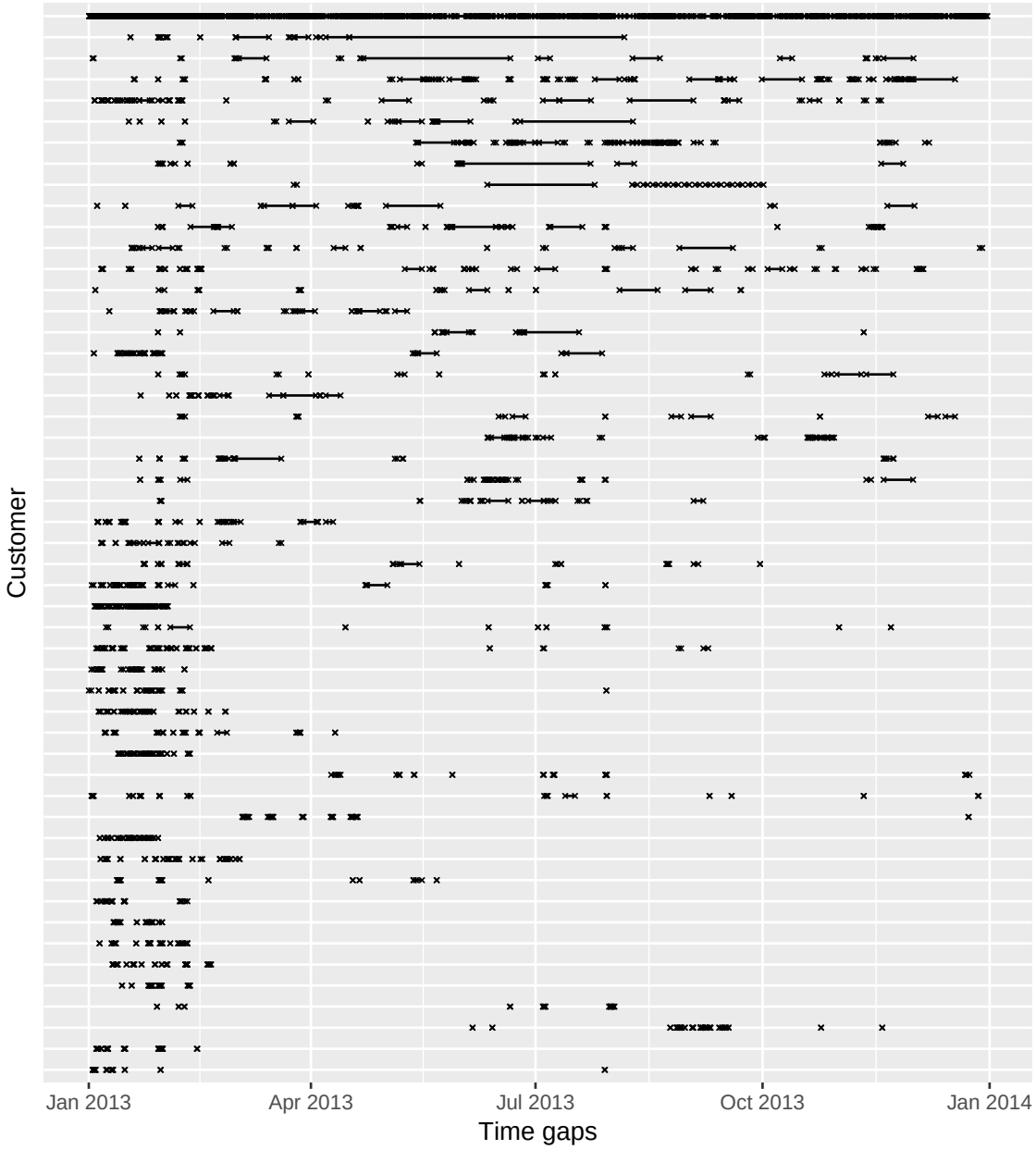

Figure 3.11: *Time gap plots for the 49 customers with most implicit missing values, and the remaining customers grouped into the one line at top. Each cross represents an observation missing in time and a line between two dots shows continuous missingness over time. Each row corresponds to one customer. Missing values occur at various times, with more in January and February than other months.*

the use of air conditioning, especially on hot days with daily average temperature greater than 25 degrees C. However, the winter time (July and August) sees two peaks in a day, which is probably due to heating in the morning and evening. This plot illustrates how the tsibble data can easily integrate with other tools and graphics.

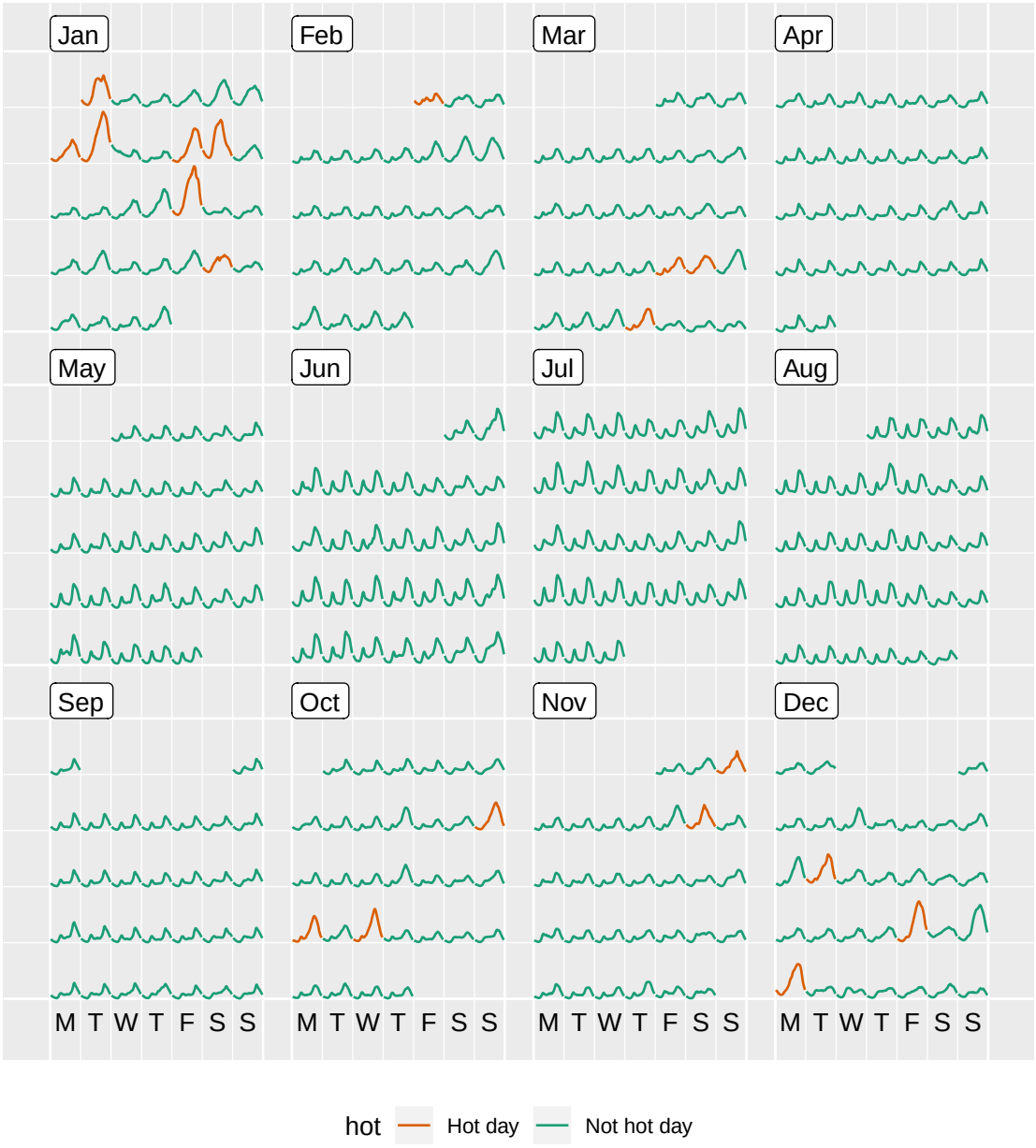

Figure 3.12: *Half-hourly average electricity use across all customers in the region, organized into calendar format, with color indicating hot days. Energy use of hot days tends to be higher, suggesting air conditioner use. Days in the winter months have a double peak suggesting heater use.*

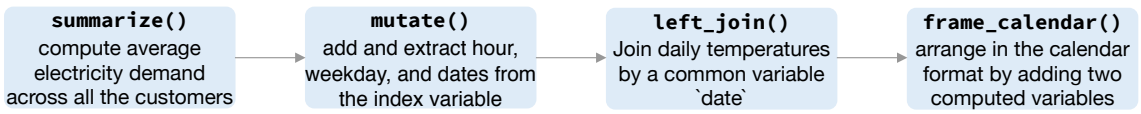

Figure 3.13: *Flow chart illustrating the pipeline that preprocessed the data for creating Figure 3.12.*

3.7 Conclusion and future work

The "tsibble" is a new data abstraction to represent temporal data, allowing the "tidy data" principles to be brought to the time domain. Tidy data begins to take shape in the state of time with the introduction of the contextual semantics of index and key. A declared index provides direct support to the time variable; variables that comprise the key define observations over time. These semantics further determine unique data entries required for a valid tsibble. No matter how temporal data arrives, a tsibble respects a time index and maintains the data richness. A tsibble frictionlessly allows transformation, visualization and modeling, and smoothly shifts between them, allowing for rapid iteration to gain data insights.

A missing piece of the **tsibble** package is to enable user-defined calendars and to respect structurally 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, which tsibble currently disregards. However, a few R packages provide functionality to create and manage many sorts of calendars, including market-specific business calendars. Generally, custom calendars are easily embedded into the tsibble framework. Consequently these tsibble operators, like `fill_gaps()`, would work out of the box, and forecasts would be generated within its definable time range.

The **tsibble** package provides the grammar of temporal data manipulation, regardless of how the data is stored. Currently, it works for managing and manipulating temporal data frames in memory locally. But it is possible to work with remote tables stored in databases, such as SQLite and MySQL, using exactly the same tsibble code. This is left for future work.

Acknowledgements

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. 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

Missingness in time: speaking the language of data

Chapter 5

Conclusion and future plans

5.1 Software development

5.2 Future work

5.3 Final words

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.

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 (visited on 09/26/2018).

Chang, W, J Cheng, J Allaire, Y Xie, and J McPherson (2018). *shiny: Web Application Framework for R*. R package version 1.1.0. https://CRAN.R-project.org/package=shiny.

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

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.

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.

Friedman, DP and M Wand (2008). *Essentials of Programming Languages, 3rd Edition*. 3rd ed. The MIT Press.

Hafen, R (2018). *geofacet: 'ggplot2' Faceting Utilities for Geographical Data*. R package version 0.1.9. https://CRAN.R-project.org/package=geofacet.

Henry, L and H Wickham (2018). *purrr: Functional Programming Tools*. R package version 0.2.5. https://CRAN.R-project.org/package=purrr.

Henry, L and H Wickham (2019a). *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 (2019b). *Tidy evaluation*. https://tidyeval.tidyverse.org.

Hofmann, H, H Wickham, and K Kafadar (2017). Letter-Value Plots: Boxplots for Large Data. *Journal of Computational and Graphical Statistics* **26**(3), 469–477.

Hyndman, RJ and G Athanasopoulos (2017). *Forecasting: Principles and Practice*. Melbourne, Australia: OTexts. OTexts.org/fpp2.

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

Jones, H (2016). *Calendar Heatmap*. Online post. https://rpubs.com/haj3/calheatmap.

Kimball, R and J Caserta (2011). *The Data WarehouseÂ 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.

Mcilroy, D, E Pinson, and B Tague (1978). Unix Time-Sharing System Forward. *The Bell System Technical Journal*, 1902–1903.

Pebesma, E (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal* **10**(1), 439–446.

Ryan, JA and JM Ulrich (2018). *xts: eXtensible Time Series*. R package version 0.11-0. https://CRAN.R-project.org/package=xts.

Sievert, C (2018). *plotly for R*. https://plotly-book.cpsievert.me.

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.

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.

Tierney, NJ and D Cook (2018). *Expanding tidy data principles to facilitate missing data exploration, visualization and assessment of imputations*. eprint: arXiv:1809.02264.

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 (2018). *wanderer4melb: Shiny App for Wandering Around the Downtown Melbourne 2016*. R package version 0.1.0. https://github.com/earowang/wanderer4melb.

Wang, E, D Cook, and R Hyndman (2019). *tsibble: Tidy Temporal Data Frames and Tools*. R package version 0.7.0. https://tsibble.tidyverts.org.

Wang, E, D Cook, and RJ Hyndman (2018a). *Calendar-based graphics for visualizing people's daily schedules*. eprint: arXiv:1810.09624.

Wang, E, D Cook, and RJ Hyndman (2018b). *sugrrants: Supporting Graphs for Analysing Time Series*. R package version 0.1.6. https://pkg.earo.me/sugrrants.

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 (2018a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. http://ggplot2.tidyverse.org, https://github.com/tidyverse/ggplot2.

Wickham, H, R François, L Henry, and K Müller (2018b). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.7. https://CRAN.R-project.org/package=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/.

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, Articles* **14**(6), 1–27.