Conversations in time: interactive visualisation to explore structured temporal data

by Earo Wang and Dianne Cook

Abstract Temporal data often has a hierarchical structure, defined by categorical variables describing different levels, such as political regions or sales products. Nesting of categorical variables produces a hierarchical structure. The **tsibbletalk** package is developed to allow a user to interactively explore temporal data, relative to the nested or crossed structures. It can help to discover differences between category levels, and uncover interesting periodic or aperiodic slices. The package implements a shared `tsibble` object that allows for linked brushing between coordinated views, and a shiny module that aids in wrapping time lines for seasonal patterns. The tools are demonstrated using two data examples: domestic tourism in Australia and pedestrian traffic in Melbourne.

Introduction

Temporal data typically arrives as a set of many observational units measured over time. Some variables may be categorical, containing a hierarchy in the collection process, that may be measurements taken in different geographic regions, or types of products sold by the one company. Exploring these multiple features can be daunting. Ensemble graphics (Unwin and Valero-Mora, 2018) bundle multiple views of a data set together into one composite figure. These provide an effective approach for exploring and digesting many different aspects of temporal data. Adding interactivity to the ensemble can greatly enhance the exploration process.

This paper describes new software, the **tsibbletalk** package, for exploring temporal data using linked views and time wrapping. We first provide some background to the approach based on setting up data structures and workflow, and give an overview of interactive systems in R. The section following introduces the **tsibbletalk** package. We explain the mechanism for constructing interactivity, to link between multiple hierarchical data objects and hence plots, and describe the set up for interactively slicing and dicing time to wrap a series on itself to investigate periodicities.

Background: tidy temporal data and workflow

The **tsibble** package (Wang et al., 2020) introduced a unified temporal data structure, referred to as a `tsibble`, to represent time series and longitudinal data in a tidy format (Wickham, 2014). A `tsibble` extends the `data.frame` and `tibble` classes with the temporal contextual metadata: `index` and `key`. The `index` declares a data column that holds time-related indices. The `key` identifies a collection of related series or panels observed over the `index`-defined period, which can comprise multiple columns. An example of a `tsibble` can be found in the monthly Australian retail trade turnover data (`aus_retail`), available in the **tsibbledata** package (O'Hara-Wild et al., 2020c), shown below. The `Month` column holds year-months as the `index`. `State` and `Industry` are the identifiers for these 152 series, which form the `key`. Note that the column `Series ID` could be an alternative option for setting up the key, but `State` and `Industry` are more readable and informative. The `index` and `key` are "sticky" columns to a `tsibble`, forming critical pieces for fluent downstream temporal data analysis.

```
#> # A tsibble: 64,532 x 5 [1M]
#> # Key:       State, Industry [152]
#>   State              Industry                  `Series ID`   Month Turnover
#>   <chr>              <chr>                     <chr>         <mth>    <dbl>
#> 1 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Apr      4.4
#> 2 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 May      3.4
#> 3 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Jun      3.6
#> 4 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Jul      4
#> 5 Australian Capital ~ Cafes, restaurants and cat~ A3349849A   1982 Aug      3.6
#> # ... with 64,527 more rows
```

In the spirit of tidy data from the **tidyverse** (Wickham et al., 2019), the **tidyverts** suite features `tsibble` as the foundational data structure, and helps to build a fluid and fluent pipeline for time series analysis. Besides **tsibble**, the **feasts** (O'Hara-Wild et al., 2020b) and **fable** (O'Hara-Wild et al.,
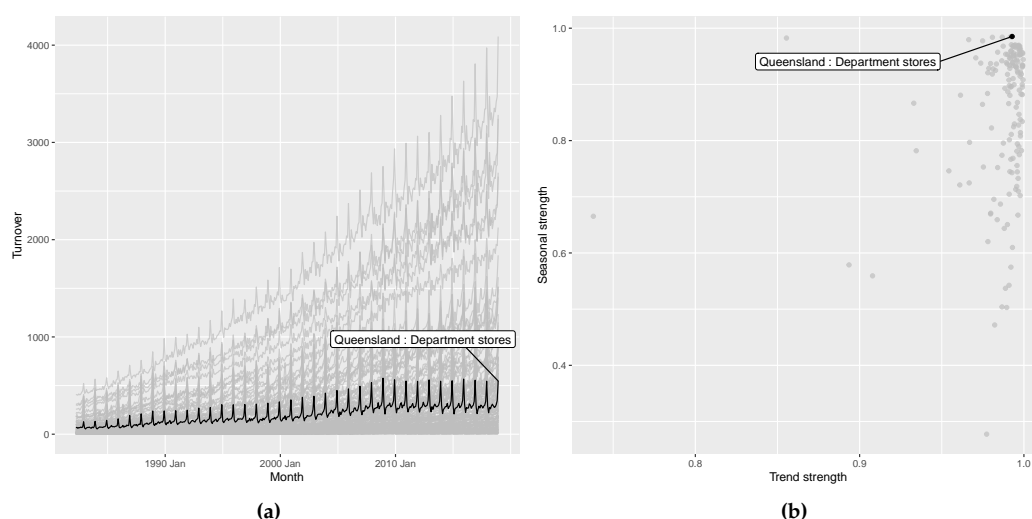
Figure 1: Plots for the `aus_retail` data, with the series of strongest seasonal strength highlighted. (a) An overlaid time series plot. (b) A scatter plot drawn from their time series features, where each dot represents a time series from (a).

2020a) packages fill the role of statistical analysis and forecasting in the **tidyverts** ecosystem. During all the steps of a time series analysis, the series of interest, denoted by the `key` variable, typically persist, through the trend modeling and also forecasting. We would typically want to examine the series across all of the keys.

Figure 1 illustrates examining temporal data with many keys. The data has 152 series corresponding to different industries in retail data. The multiple series are displayed using an overlaid time series plot, along with a scatterplot of two variables (trend versus seasonal strength) from feature space, where each series is represented by a dot. The feature space is computed using the `features()` function from **feasts**, which summarises the original data for each series using various statistical features. This function along with other **tidyverts** functions is `tsibble`-aware, and outputs a table in a reduced form where each row corresponds to a series, which can be graphically displayed as in Figure 1b.

Figure 1 has also been highlighted to focus on the one series with the strongest seasonality. To create this highlighting, one needs to first filter the interesting series from the features table, and join back to the original `tsibble` in order to examine its trend in relation to others. This procedure can soon grow cumbersome if many series are to be explored. It illustrates a need to query interesting series on the fly. Although these two plots are static, we can consider them as linked views because the common key variables link between the two data tables producing the two plots. This motivates the work in this package, described in this paper, to enable interactivity of `tsibble` and `tsibble`-derived objects for rapid exploratory data analysis.

Overview of interactivity

There is a long history of interactive data visualisation research and corresponding systems. Within R, the systems can be roughly divided into systems utilising web technology and those that don't.

R **shiny** (Chang et al., 2020) and **htmlwidgets** (Vaidyanathan et al., 2019) provide infrastructure connecting R with HTML elements and JavaScript that support the interactivity. The **htmlwidgets** package makes it possible to embed JavaScript libraries into R so that users are able to write only R code to generate web-based plots. Many JavaScript charting libraries have been ported to R as HTML widgets, including **plotly** (Sievert, 2020), **rbokeh** (Hafen and Continuum Analytics, Inc., 2020), and **leaflet** (Cheng et al., 2019) for maps. Interactions between different widgets can be achieved with **shiny** or **crosstalk** (Cheng, 2020). The **crosstalk** extends **htmlwidgets** with shared R6 instances to support linked brushing and filtering across widgets, without relying on **shiny**.

Systems without the web technology include **grDevices**, **loon** (Waddell and Oldford, 2020), based on Tcl/Tk, and **cranvas** (Xie et al.) based on Qt. They offer a wide array of pre-defined interactions, such as selecting and zooming, to manipulate plots via mouse action, keyboard strokes, and menus. The **cranvastime** package (Cheng et al.) is an add-on to **cranvas**, which provides specialised interactions

for temporal data, such as wrapping and mirroring.

The techniques implemented in work described in this paper utilise web technology, including **crosstalk**, **plotly**, and R **shiny**.

Using a shared temporal data object for interactivity

The **tsibbletalk** package introduces a shared tsibble instance built on a `tsibble`. This allows for seamless communication between different plots of temporal data. The as_shared_tsibble() function turns a `tsibble` into a shared instance, SharedTsibbleData, which is a subclass of SharedData from **crosstalk**. This is an R6 object driving data transmission across multiple views, due to its mutable and lightweight properties. The **tsibbletalk** package aims to streamline interactive exploration of temporal data, with the focus of temporal elements and structured linking.

Linking between plots

As opposed to one-to-one linking, **tsibbletalk** defaults to categorical variable linking where selecting one or more observations in one category will broadcast to all other observations in this category. That is, linking is by key variables: within the time series plot, click on any data point, and whole line will be highlighted in response. The as_shared_tsibble() uses `tsibble`'s key variables to achieve these types of linking.

The approach can also accommodate temporal data of nesting and crossing structures. These time series are referred to as hierarchical and grouped time series in the literature (Hyndman and Athanasopoulos, 2017). The `aus_retail` above is an example of grouped time series. Each series in the data corresponds to all possible combinations of the `State` and `Industry` variables, which means they are intrinsically crossed with each other. When one key variable is nested within another, such as regional areas within a state, this is considered to be a hierarchical structure.

The spec argument in as_shared_tsibble() provides a means to construct hybrid linking, that incorporates hierarchical and categorical linking. A symbolic formula can be passed to the spec argument, to define the crossing and/or nesting relationships among the key variables. Adopting Wilkinson and Rogers (1973)'s notation for factorial models, the spec follows the / and * operator conventions to declare nesting and crossing variables, respectively. The spec for the `aus_retail` data is therefore specified as `State * Industry` or `Industry * State`, which is the default for the presence of multiple key variables. If there is a hierarchy in the data, using / is required to indicate the parent-child relation, for a strictly one directional `parent/child`.

To illustrate nesting and crossing we use the `tourism_monthly` dataset (Tourism Research Australia, 2020) packaged in **tsibbletalk**. It contains monthly domestic overnight trips across Australia. The key is comprised of three identifying variables: `State`, `Region`, and `Purpose` (of trip), in particular `State` nesting of `Region`, together crossed with `Purpose`. This specification can be translated as follows:

```
library(tsibbletalk)
tourism_shared <- tourism_monthly %>%
  as_shared_tsibble(spec = (State / Region) * Purpose)
```

There is a three-level hierarchy: the root node is implicitly Australia, and geographically disaggregated to states and lower-level tourism regions. A new handy function plotly_key_tree() has been implemented to help explore the hierarchy. It interprets hierarchies in the shared tsibble's spec as a tree view, built with **plotly**. The following code line produces the linked tree diagram (left panel of Figure 2). The visual for the tree hierarchy detangles a group of related series and provides a bird's eye view of the data organisation.

```
p_l <- plotly_key_tree(tourism_shared, height = 1100, width = 800)
```

The tree plot provides the graphics skeleton, upon which the rest of data plots can be attached. In this example, small multiples of line plots are placed at the top right of Figure 2 to explore the temporal trend across regions, by trip purpose. The shared tsibble data can be directly piped into **ggplot2** code to create this.

```
library(ggplot2)
p_tr <- tourism_shared %>%
  ggplot(aes(x = Month, y = Trips)) +
  geom_line(aes(group = Region), alpha = .5, size = .4) +
  facet_wrap(~ Purpose, scales = "free_y") +
  scale_x_yearmonth(date_breaks = "5 years", date_labels = "%Y")
```
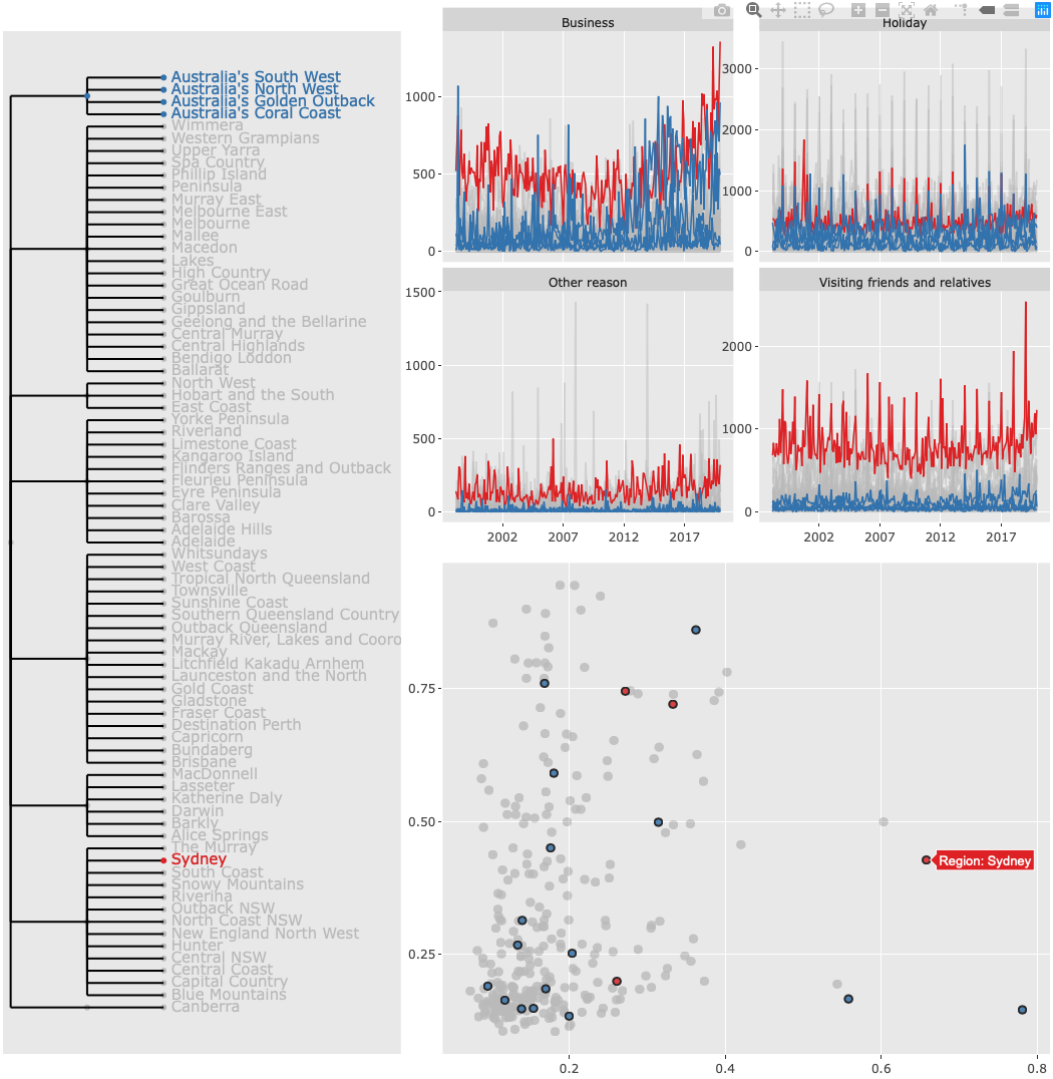
Figure 2: Snapshot of exploring an ensemble of linked plots of the Australian tourism data, built on a `tourism_shared` object. It also illustrates persistent linked brushing to compare two groups.

These line plots are heavily overplotted. To tease apart structure in the multiple time series, the `features()` function computes interesting characteristics, including the measures of trend and seasonality. These are displayed in the scatterplot at bottom right, where one dot represents one series.

```
library(feasts)
tourism_feat <- tourism_shared %>%
  features(Trips, feat_stl)
p_br <- tourism_feat %>%
  ggplot(aes(x = trend_strength, y = seasonal_strength_year)) +
  geom_point(aes(group = Region), alpha = .8, size = 2)
```

There is one final step, to compose the three plots into an ensemble of coordinated views for exploration, shown in Figure 2. (This is the interactive realisation of Figure 1).

```
library(plotly)
subplot(p_l,
  subplot(
    ggplotly(p_tr, tooltip = "Region", width = 1100),
    ggplotly(p_br, tooltip = "Region", width = 1100),
    nrows = 2),
  widths = c(.4, .6)) %>%
  highlight(dynamic = TRUE)
```

Since all plots are created from one shared tsibble data source, they are self-linking views. Nodes, lines, and points are hoverable and clickable. Given the spec, clicking either one element in any plot highlights all points that match the `Region` category, that is, categorical linking. Figure 2 is a static view of an interactive exploration. The steps in getting to this point were:

1. A branch of the tree corresponding to Western Australia was first selected. (The names of the regions are a little odd, which is a quirk of the data set, but all four areas, Australia's South West, ..., correspond to tourist destinations in Western Australia. Hovering over the node on the branch brings up the state name.) This generated the response in the line plots and the scatterplot that coloured corresponding time series and points as blue.
2. To enable persistent selection, in oder to compare regions or states, "Shift" and click on the tree was done, after switching the colour to red. This generated the response that points and time series corresponding to Sydney were highlighted in red.
3. Hovering over the points brings up the label for Sydney.

The domestic tourism sees Sydney as one of the most popular destinations in realm of business and friends visiting over the years. Despite the relatively weaker performance in Western Australia, Australia's North West region sees a strongest upward trend in Business, bypassing Sydney in some years.

In summary, shared tsibble data nicely bridges between the **crosstalk** and **tidyverts** ecosystems for temporal data using the common "key". The `as_shared_tsibble()` provides a symbolic user interface for effortless construction of a hybrid of hierarchical and categorical linking between plots. The `plotly_key_tree()` function, in turn, decodes the hierarchical specification to plot a tree for data overview and navigation, when accompanied by more detailed plots.

Slicing and dicing time

An important aspect of temporal data is the time context. Time has cyclical structure, that may correspond to seasonal patterns to be discovered. The `index` component of the (shared) tsibble data forms the basis for exploring seasonality. To investigate for periodic or aperiodic patterns, series should be wrapped on themselves, where the index is broken into temporal components like quarter or day. We shall explore this with pedestrian traffic in Melbourne, Australia.

The city of Melbourne has sensors installed at various locations, to record hourly counts of pedestrians, in order to capture daily rhythms of the downtown (City of Melbourne, 2020). Figure 3 shows the first five months of 2020 foot traffic at four different locations, for three different time slices, daily, weekly and full five months. Plot 3a shows hourly counts from January to May on an absolute time line, facetted by locations. The stage 3 COVID-19 lockdown, on March 16, is marked by a change of colour. (The pre-lockdown period is coloured with dark green and lockdown with orange.) We can see a significant decline in foot traffic at all four locations. QV Market is less affected probably because this is a major produce market, an essential service that continued to operate. Bourke St, a major shopping center, sees a gradual uptick in the last weeks of the period indicating that people were getting back into the shops.
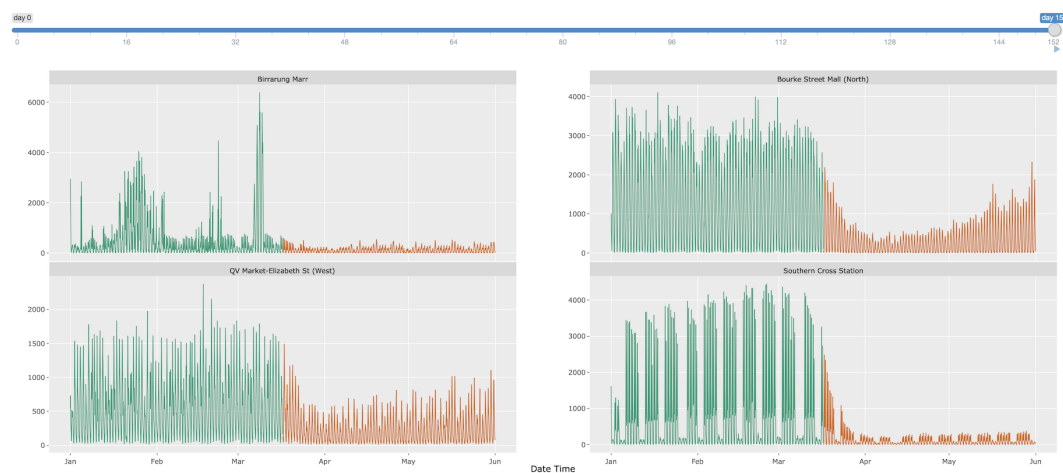
(a) Initial overview state
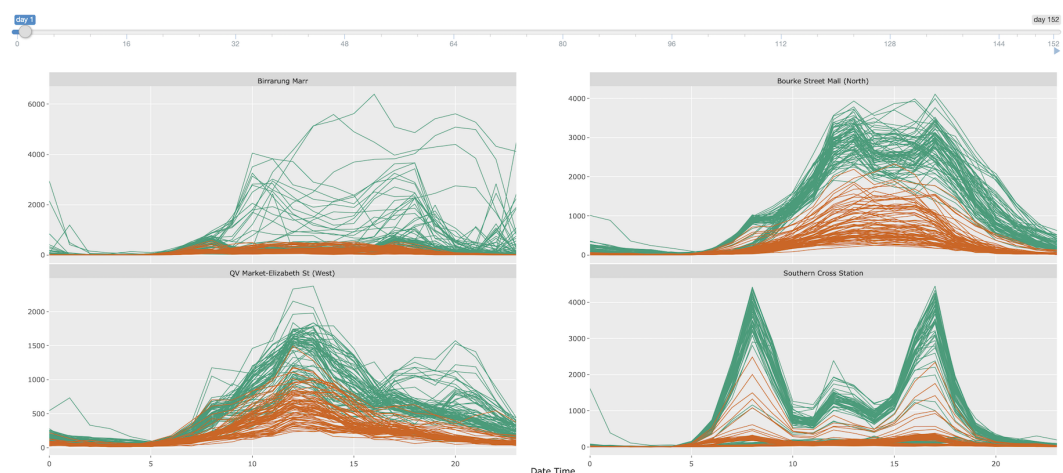
(b) 1-day state

(c) 7-day state, anchoring to Monday

Figure 3: Snapshots wrapping after slicing the `pedestrian20` data at different intervals, (a) none, (b) daily and (c) weekly. This type of interaction is made possible with Shiny elements.

Figure 3b and 3c show slicing and wrapping of the series into daily and weekly sections, respectively. Multiple seasonalities pop out. There tends to be a daily pattern, especially visible at the main train station, Southern Cross Station. There is also a weekday vs weekend pattern, also most visible at Southern Cross Station. These seasonal patterns are still present during the lockdown, but the magnitude is greatly reduced. Numbers are also down at the produce market and the shopping center. Birrarung Marr is the most affected. This is the location of special events, and it is clear that these have completely disappeared during lockdown.

The wrapping procedure involves slicing the time index into seasonal periods of interest, and the result is diced time. For example, hourly pedestrian data can be decomposed into 24-hour blocks, which then overlays the counts for all respective days, as done in plot 3b. For exploration, this slice position should be controlled interactively, so that many different slices can be examined rapidly. This can be achieved using shiny, with the functions provided in the **tsibbletalk**.

This shiny module, decoupled to `tsibbleWrapUI()` and `tsibbleWrapServer()`, presents a clean interface and forms a reusable component that could be embedded in any shiny application. In general, a shiny module provides a vehicle for modularising shiny applications, relevant for both users and developers. As with all shiny modules, the first argument in both functions in **tsibbletalk** requires a user-supplied id string that must be unique. The UI function `tsibbleWrapUI()` simply shows a slider that animates or controls the number of periods to be diced. The workhorse is the server function `tsibbleWrapServer()`, encapsulating the algorithm that transforms data and sends messages to update the plot accordingly. The `plot` argument expects a `ggplot` or `plotly` object, where one can plot data using either lines or other graphical elements (such as boxplots). As the function name suggests, a (shared) tsibble is needed to start the engine, so that the time `index` can be retrieved for dissection. The `period` option semantically takes a desired number of seasonal periods to be shifted, for example data shifted by "1 day", "2 days", or "1 week", etc. In other words, the `period` defines the grind level. For date-times (represented by `POSIXt`), the granularity ranges from fine "day" to a much coarser "year". The following code snippet generates Figure 3.

```
library(shiny)
p_line <- pedestrian20 %>%
  ggplot(aes(x = Date_Time, y = Count, colour = Lockdown)) +
  geom_line(size = .3) +
  facet_wrap(~ Sensor, scales = "free_y") +
  labs(x = "Date Time") +
  scale_colour_brewer(palette = "Dark2") +
  theme(legend.position = "none")

ui <- fluidPage(
  tsibbleWrapUI("dice")
)
server <- function(input, output, session) {
  tsibbleWrapServer("dice", ggplotly(p_line, height = 700), period = "1 day")
}
shinyApp(ui, server)
```

Figure 3a corresponds to the initial state, with the slider incremented by 1-day units. The "play" button near the end of slider can automatically animate the slicing and dicing process, walking the viewer through all 24 hours of the 152 days. Alternatively, users can drag the slider to examine selected slices.

In response to the slider input, the plot will be updated and loaded with newly transformed data. At its core, keeping the application as performant as possible is the top priority. Without completely redrawing the plot, the `plotlyProxy()` react method is invoked internally for talking to shiny. The underlying tsibble data is being called back and processed in R. Only transformed data gets fed back to the shiny server, for updating with resetting the x-axis ranges and breaks. The other plot configurations, such as marks, y-axes, and layouts, are cached and used as is.

The new shiny module exploits the temporal aspect for a tsibble object, available through the `index` attribute. It allows users to slide through relative periods to digest seasonal behaviours, with nimble user experience.

Summary

At the heart of the **tsibbletalk** package is a blending of the best bits from **tsibble**, **crosstalk**, **plotly**, and **shiny**.

The `as_shared_tsibble()` turns a `tsibble` object to a shared data class, with an option to express any nesting and crossing structures from the `key` attribute. If nesting is found in the data, the `plotly_key_tree()` creates an interactive hierarchical tree to help with data overview. This sets the stage for hierarchical and categorical linking between multiple views from one shared tsibble.

A new shiny module, `tsibbleWrapUI()` and `tsibbleWrapServer()`, provides a lens for looking at temporal aspects of a `tsibble`, in particular seasonal or cyclical variations. The slicing and dicing technique efficiently wrap time lines for user-defined plots. The `plotlyProxy()` react method makes it possible to send wrapped data to the server and amend the plot straight way.

Bibliography

W. Chang, J. Cheng, J. Allaire, Y. Xie, and J. McPherson. *shiny: Web Application Framework for R*, 2020. URL https://CRAN.R-project.org/package=shiny. R package version 1.5.0. [p2]

J. Cheng. *crosstalk: Inter-Widget Interactivity for HTML Widgets*, 2020. URL https://CRAN.R-project.org/package=crosstalk. R package version 1.1.0.1. [p2]

J. Cheng, B. Karambelkar, and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2019. URL https://CRAN.R-project.org/package=leaflet. R package version 2.0.3. [p2]

X. Cheng, D. Cook, and H. Hofmann. Enabling interactivity on displays of multivariate time series and longitudinal data. 25(4):1057–1076. ISSN 1061-8600, 1537-2715. URL https://www.tandfonline.com/doi/full/10.1080/10618600.2015.1105749. [p2]

City of Melbourne. *Pedestrian Volume in Melbourne*, 2020. URL http://www.pedestrian.melbourne.vic.gov.au. [p5]

R. Hafen and Continuum Analytics, Inc. *rbokeh: R Interface for Bokeh*, 2020. URL https://CRAN.R-project.org/package=rbokeh. R package version 0.5.1. [p2]

R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Melbourne, Australia, 2017. URL OTexts.org/fpp2. [p3]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *fable: Forecasting Models for Tidy Time Series*, 2020a. URL https://CRAN.R-project.org/package=fable. R package version 0.2.1. [p1]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *feasts: Feature Extraction and Statistics for Time Series*, 2020b. URL https://CRAN.R-project.org/package=feasts. R package version 0.1.5. [p1]

M. O'Hara-Wild, R. Hyndman, and E. Wang. *tsibbledata: Diverse Datasets for 'tsibble'*, 2020c. URL https://CRAN.R-project.org/package=tsibbledata. R package version 0.2.0. [p1]

C. Sievert. *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020. ISBN 9781138331457. URL https://plotly-r.com. [p2]

Tourism Research Australia. *Australian domestic overnight trips*, 2020. URL https://www.tra.gov.au. [p3]

A. Unwin and P. Valero-Mora. Ensemble Graphics. *Journal of Computational and Graphical Statistics*, 27 (1):157–165, 2018. ISSN 1061-8600, 1537-2715. URL https://www.tandfonline.com/doi/full/10.1080/10618600.2017.1383264. [p1]

R. Vaidyanathan, Y. Xie, J. Allaire, J. Cheng, and K. Russell. *htmlwidgets: HTML Widgets for R*, 2019. URL https://CRAN.R-project.org/package=htmlwidgets. R package version 1.5.1. [p2]

A. Waddell and R. W. Oldford. *loon: Interactive Statistical Data Visualization*, 2020. URL https://CRAN.R-project.org/package=loon. R package version 1.3.1. [p2]

E. Wang, D. Cook, and R. J. Hyndman. A new tidy data structure to support exploration and modeling of temporal data. *Journal of Computational and Graphical Statistics*, 29(3):466–478, 2020. doi: 10.1080/10618600.2019.1695624. [p1]

H. Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. [p1]

H. Wickham, M. Averick, J. Bryan, W. Chang, L. D. McGowan, R. François, G. Grolemund, A. Hayes, L. Henry, J. Hester, M. Kuhn, T. L. Pedersen, E. Miller, S. M. Bache, K. Müller, J. Ooms, D. Robinson, D. P. Seidel, V. Spinu, K. Takahashi, D. Vaughan, C. Wilke, K. Woo, and H. Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019. URL https://doi.org/10.21105/joss.01686. [p1]

G. N. Wilkinson and C. E. Rogers. Symbolic description of factorial models for analysis of variance. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 22(3):392–399, 1973. ISSN 00359254, 14679876. URL http://www.jstor.org/stable/2346786. [p3]

Y. Xie, H. Hofmann, and X. Cheng. Reactive programming for interactive graphics. 29(2):201–213. ISSN 0883-4237. URL http://projecteuclid.org/euclid.ss/1408368571. [p2]

Earo Wang
The University of Auckland
Department of Statistics

earo.wang@auckland.ac.nz

Dianne Cook
Monash University
Department of Econometrics and Business Statistics

dicook@monash.edu