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

Earo Wang
Department of Econometrics and Business Statistics,
Monash University, VIC 3800
Australia.
Email: earo.wang@monash.edu
Corresponding author

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

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

29 January 2019

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

Abstract

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**.

Keywords: time series, data wrangling, tidy data, R, forecasting, data science, exploratory data analysis, data pipelines

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.

r4ds proposed the tidy data workflow, which provides a conceptual framework for processing data (as described in Figure 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.

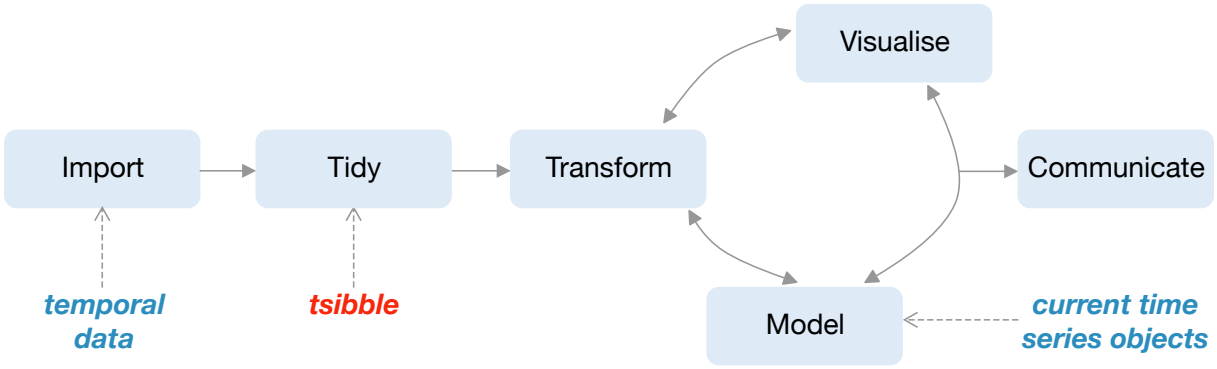

Figure 1: *Illustration of the data science workflow, drawn from* **r4ds**, *showing how current time series tools interface with the workflow and how the tsibble structure and tools integrate. The new data structure, tsibble, makes the connection between temporal data input, and specialist modeling formats. It provides elements at the "tidy" step, which produce tidy temporal data for time series visualization and modeling.*

The paper is structured as follows. Section 2 reviews temporal data structures corresponding to time series and longitudinal analysis, and discusses "tidy data" and the grammar of data manipulation. Section 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 4, followed by a discussion of the design choices made in the software structure in Section 5. Two case studies are presented in Section 6 illustrating temporal data exploration using the newly implemented infrastructure. Section 7 discusses future work.

2 Data structures

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.

2.2 Tidy data and the grammar of data manipulation

wickham2014tidy 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** (**R-dplyr**), 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 (**RJ-2018-009**) and missing value handling in the **naniar** package (**tierney-naniar-2018**). Temporal data tools need to catch up.

2.3 Existing time series standards in R

Current standards, provided by the native `ts` object in R, and extended by **zoo** (**zoo2005**) and **xts** (**R-xts**), assemble temporal data into matrices with implicit time indexes. These objects were designed for modeling methods. The diagram in the style of Figure 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** (**R-tibbletime**) 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 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 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 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 2.

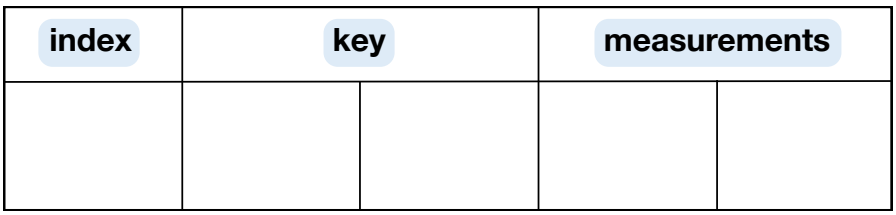

Figure 2: *The architecture of the tsibble structure is built on top of the data frame—a two-dimensional array in R, with time series semantics: index and key.*

To demonstrate the concept of the tsibble, Table 1 presents a subset of tuberculosis cases estimated by **tb-data**. 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.

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.

Table 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

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.

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_relational_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 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 (**fpp**). 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 4.3.

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.

4 Data pipelines

The tsibble structure and operations support data pipelines for sequencing analysis. There has been a long history of pipeline discussions and implementation centering around data in various aspects. A data pipeline describes the general flow of data through an analysis, and can generally assist in conceptualizing the process as it might be applied to a variety of problems. The Extract, Transform, and Load (ETL) process from recent data warehousing literature dating back to **kimball2011data** outlines the workflow to prepare data for analysis, and can be considered a data pipeline. Building a data pipeline can be technically difficult to make it sufficiently general for a variety of data, with many implementation decisions to be made about the interface, input and output objects and functionality. It is useful to articulate the data pipeline induced by new data tools.

unix 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. This shapes the Unix toolbox philosophy: "each do one simple thing, do it well, and most importantly, work well with each other" (**unix-philosophy**).

viewing-pipeline describes a viewing pipeline for interactive statistical graphics, that takes control of the transformation from data to plot. **xgobi**, **ggobi**, **orca**, **plumbing** and **xie2014reactive** 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. **xie2014reactive** 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.

The tidy data abstraction lays a pipeline infrastructure 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 and fluid 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.

4.1 Time-based transformations

The time-based pipeline shepherds raw temporal data through to time series analysis, and plots. It is possible and recommended to check for identical entries of key and index before analysis. Duplicates signal the data quality issue, which would likely affect succeeding 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. Figure 3 illustrates the time-based pipeline.

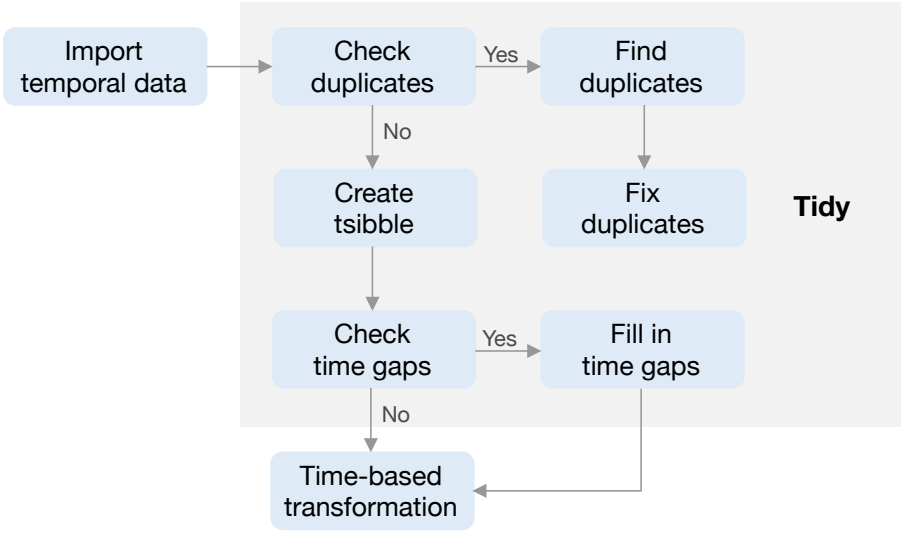

Figure 3: *A time series pipeline is different from a regular data pipeline: (1) check if the key and index uniquely identify each observation in the data whist creating a tsibble; (2) check if there are time gaps in the tsibble before analysis.*

A suite of verbs are used to flatten the lumpy path from temporal data to an object that can be directly modeled in the tsibble framework, and transformed for visualization. Many time operations like lag/lead and time series models, assume an intact vector input ordered in time. To prevent inviting these errors into the analysis, it is a good practice to check and inspect any time gaps of the tsibble in the first place. Several tools are provided to understand and tackle implicit missing values in: (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 data; (4) `fill_gaps()` turns them into explicit ones, along with imputing by values or functions.

A tsibble is an object, conceptually considered as a noun, and hence an action performed on the object can be phrased as a verb. The principle that underpins most verbs is a tsibble in and a tsibble out, thereby striving to retain a valid tsibble by automating index and key updates under the hood. Attention has been paid to error handling. If a tsibble cannot be maintained in the output of a pipeline module, for example the index is dropped by aggregation, an error informs users of the problem and suggests alternatives. This avoids surprising users and reminds them of the time context.

Table 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

Each verb focuses on one operation to achieve its goal. The verb should be self-explanatory to advise what it is supposed to do or fail, for example `filter()` picks observations, `select()` picks variables, and `left_join()` joins two tables. These general-purpose verbs are made

available in the **tidyverse** suite. When used in the temporal context, these verbs are adapted to the time domain. A perceivable difference is summarizing variables between data frame and tsibble. The former will reduce to a single summary, whereas the latter will obtain the index and their corresponding summaries. New tsibble-specific verbs (Table 2) are provided to expand the **tidyverse** vocabulary. Users who are already familiar with **tidyverse**, will experience a gentle learning curve for mastering tsibble verbs and glide into temporal data analysis with low cognitive load.

eopl 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 (**R-magrittr**). 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.

4.2 Time series visualization

As a special case of the data frame, a tsibble pipes directly into the grammar of graphics. It is easy to create and extend specialist time-related plotting methods based on tsibble structure, such as autocorrelation plots and calendar-based graphics (**wang-calendarviz-2018**).

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 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.

5 Software structure and design decisions

5.1 Data first

The primary force that drives the software's design choices is "data". All functions in the package **tsibble** (**R-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 1, contains (1) data dimension with its shorthand time interval, alongside time zone if date-time, (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
```

5.2 Functional programming

Rolling window calculations are widely used techniques in time series analysis, but are not restricted to temporal applications. These operations are dependent on having an ordering, particularly time ordering for temporal data. Three common types of variations for sliding window operations are:

1. **slide**: sliding window with overlapping observations.

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

Figure 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.

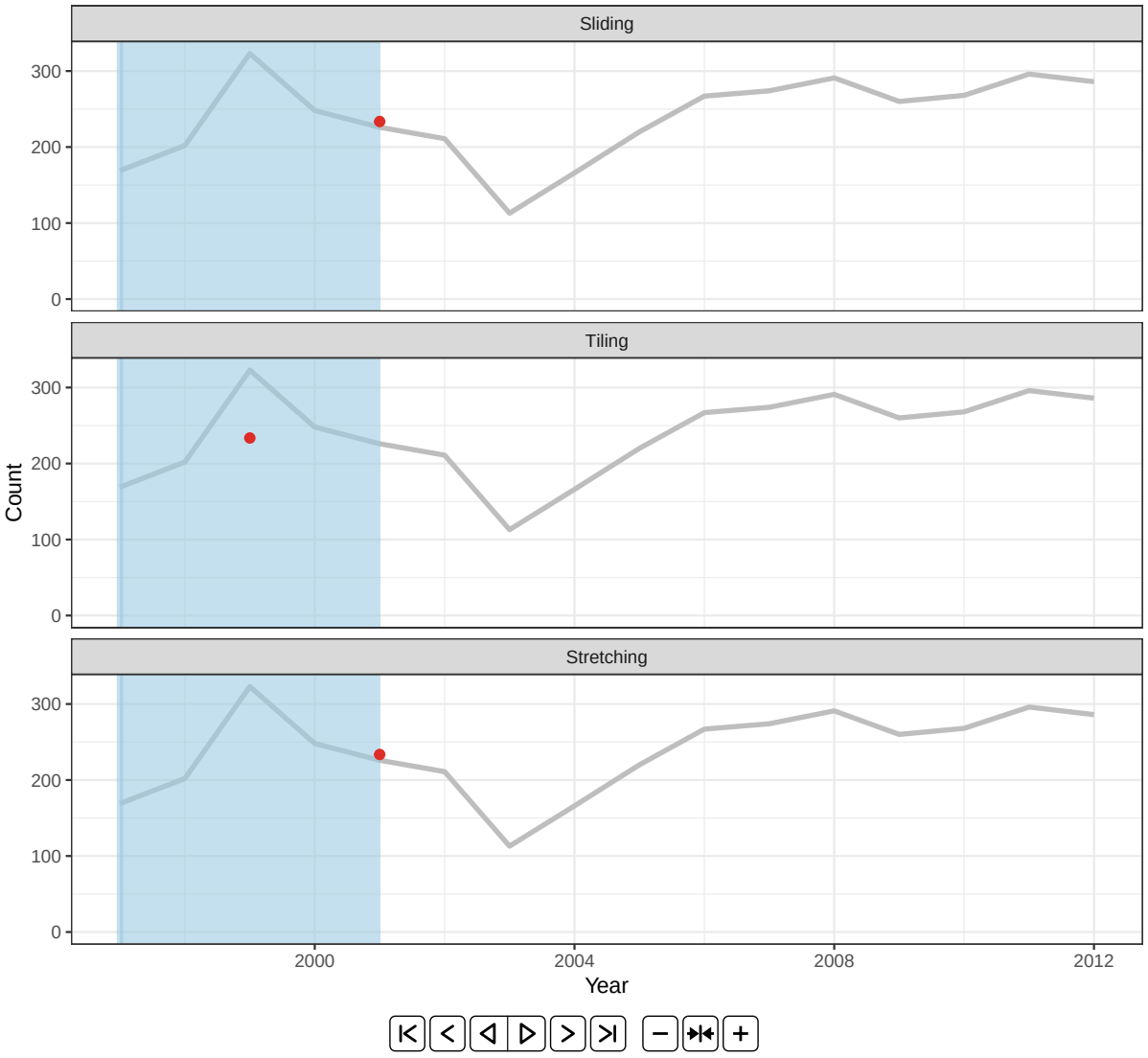

Figure 4: *An illustration of window of size 5 rolling over annual tuberculosis cases in Australia. (Animation needs to be viewed with Adobe Acrobat Reader.)*

Rolling window uses a programming paradigm—functional programming, which is different from those table verbs listed in Table 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 (**R-purrr**) 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 doesn't just iterate but rolls over a sequence of elements, with `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 (**R-futureR-furrr**). This family of functions empowers users to incorporate window-related operations in their workflows.

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. When collapsing a tsibble to a less granular interval, these atomic functions can be combined with the `index_by()` table verb. A different function results in easily switching to aggregation of different time resolution. Since these 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 current tsibble zone. It is generally easier to trace back the errors users encounter from separating 1d and 2d functions.

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 rely on S3 classes and methods in R (**adv-r**). 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** (**R-nanotime**), 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()`, `count_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 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 needed.

6 Case studies

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 (**flights-data**). 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 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.

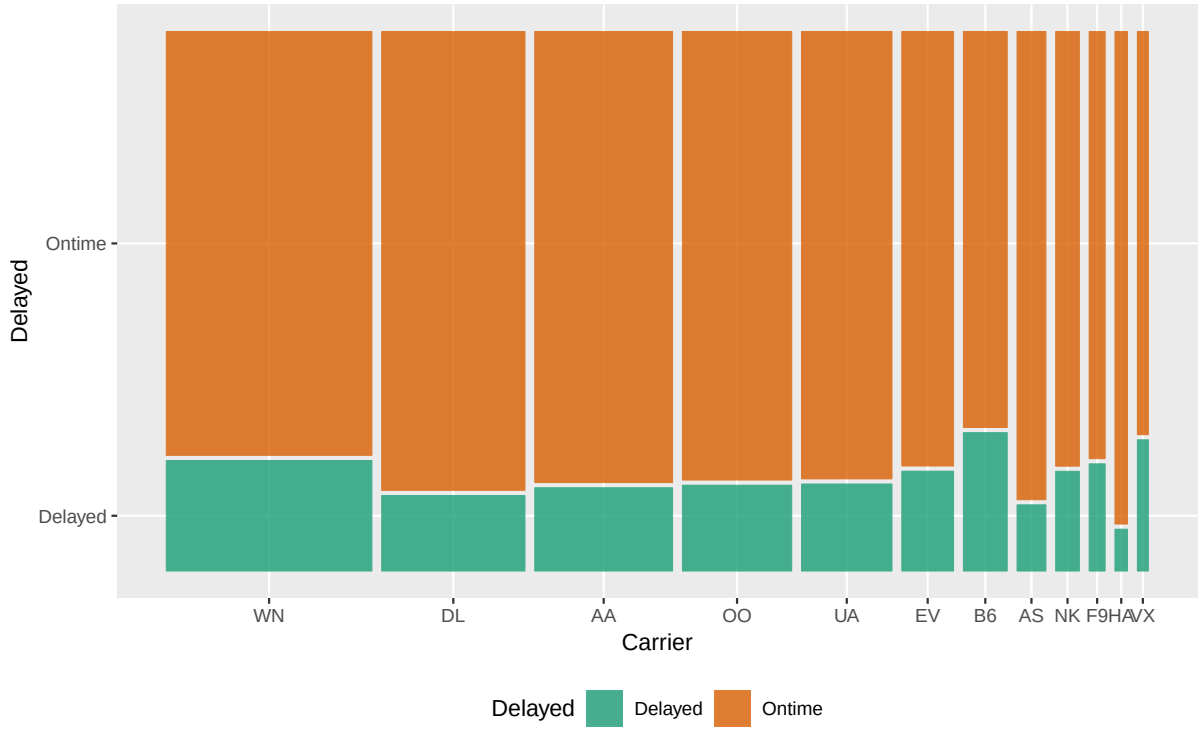

Figure 5: *Mosaic plot showing the association between the size of airline carriers and the delayed proportion of departures in 2017. Southwest Airlines is the largest operator, but does not operate as efficiently as Delta. Hawaiian Airlines, a small operator, outperforms the rest.*

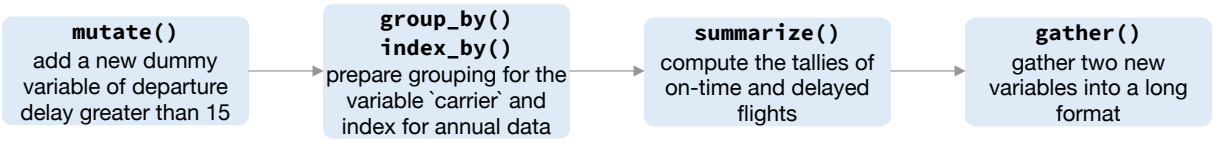

Figure 6: *Flow chart illustrating the pipeline that preprocessed the data for creating Figure 5.*

Figure 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 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. 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 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 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).

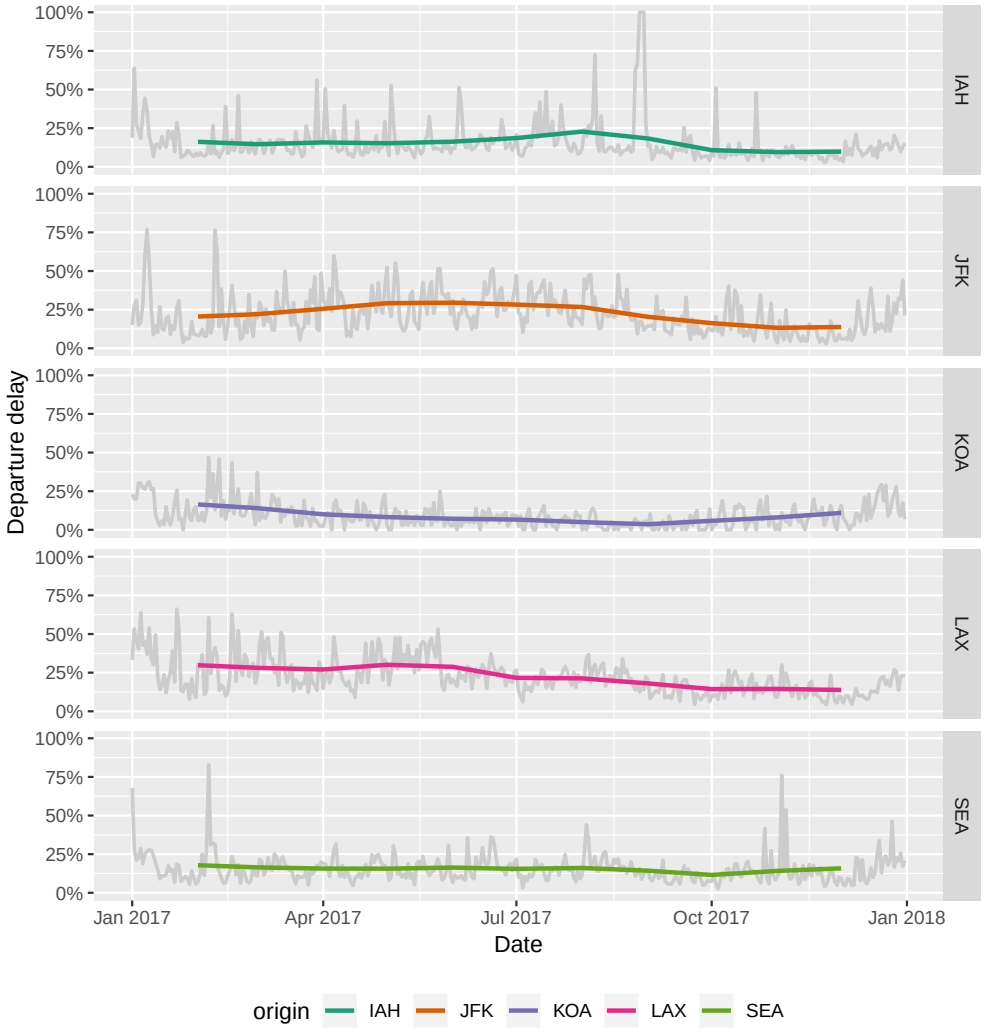

Figure 7: *Daily delayed percentages for departure with two-month moving averages overlaid at five international airports. There are least fluctuations and relatively fewer delays observed at KOA airport. The estimates of temporal trend are around 25% across the other four airports, but highlight different time periods of severe delays.*

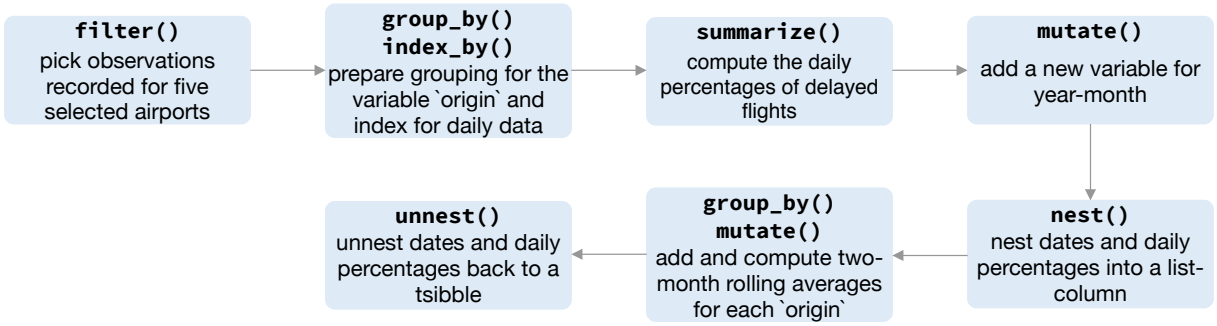

Figure 8: *Flow chart illustrating the pipeline that preprocessed the data for creating Figure 7.*

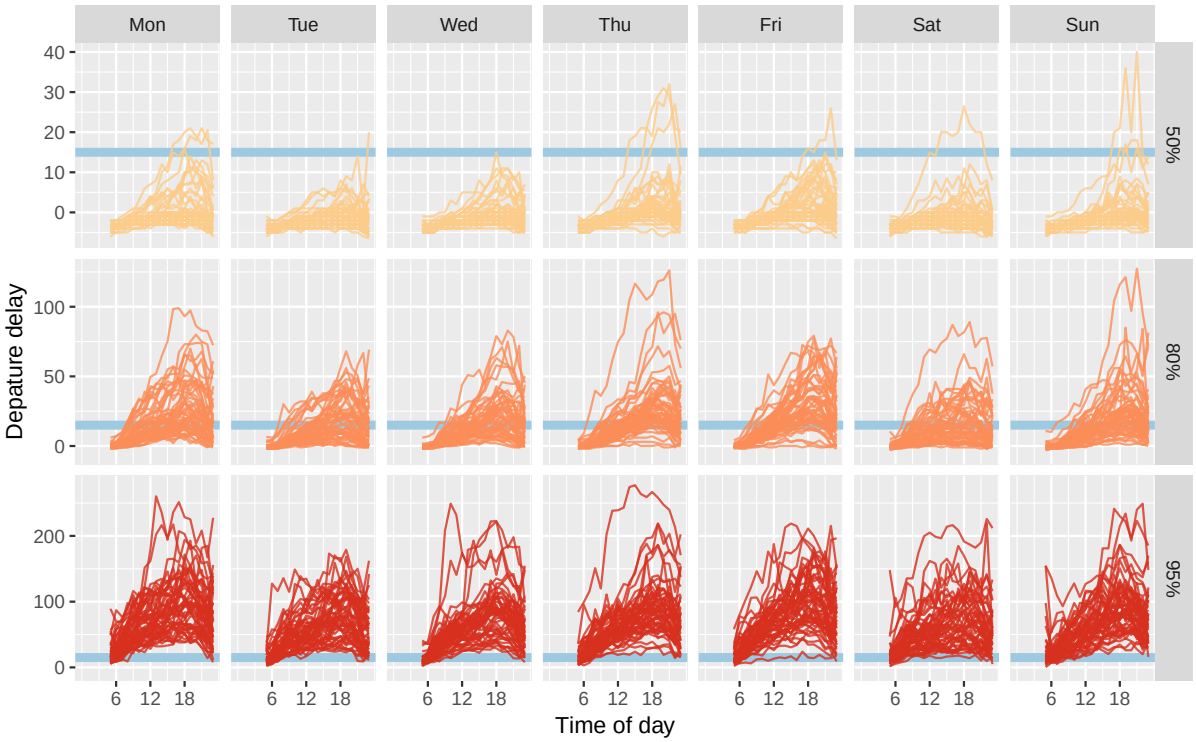

Figure 9: *Small multiples of lines about departure delay against time of day, faceting day of week and 50%, 80% and 95% quantiles. A blue horizontal line indicates the 15-minute on-time standard to help grasp the delay severity. Passengers are apt to hold up around 18 during a day, and are recommended to travel early. The variations increase substantially as the upper tails.*

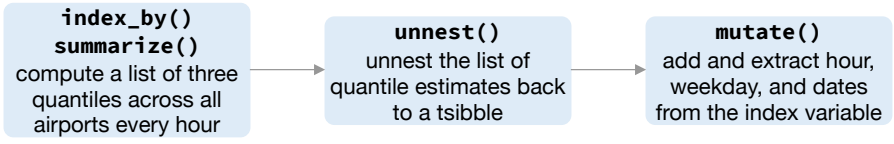

Figure 10: *Flow chart illustrates the pipeline that preprocesses the data for creating Figure 9.*

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 (**smart-meter**). 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. Customers' demographic data provides explanatory variables other than time in a different data table. Two data tables might be joined to explore different sources that contribute to 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 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 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 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.

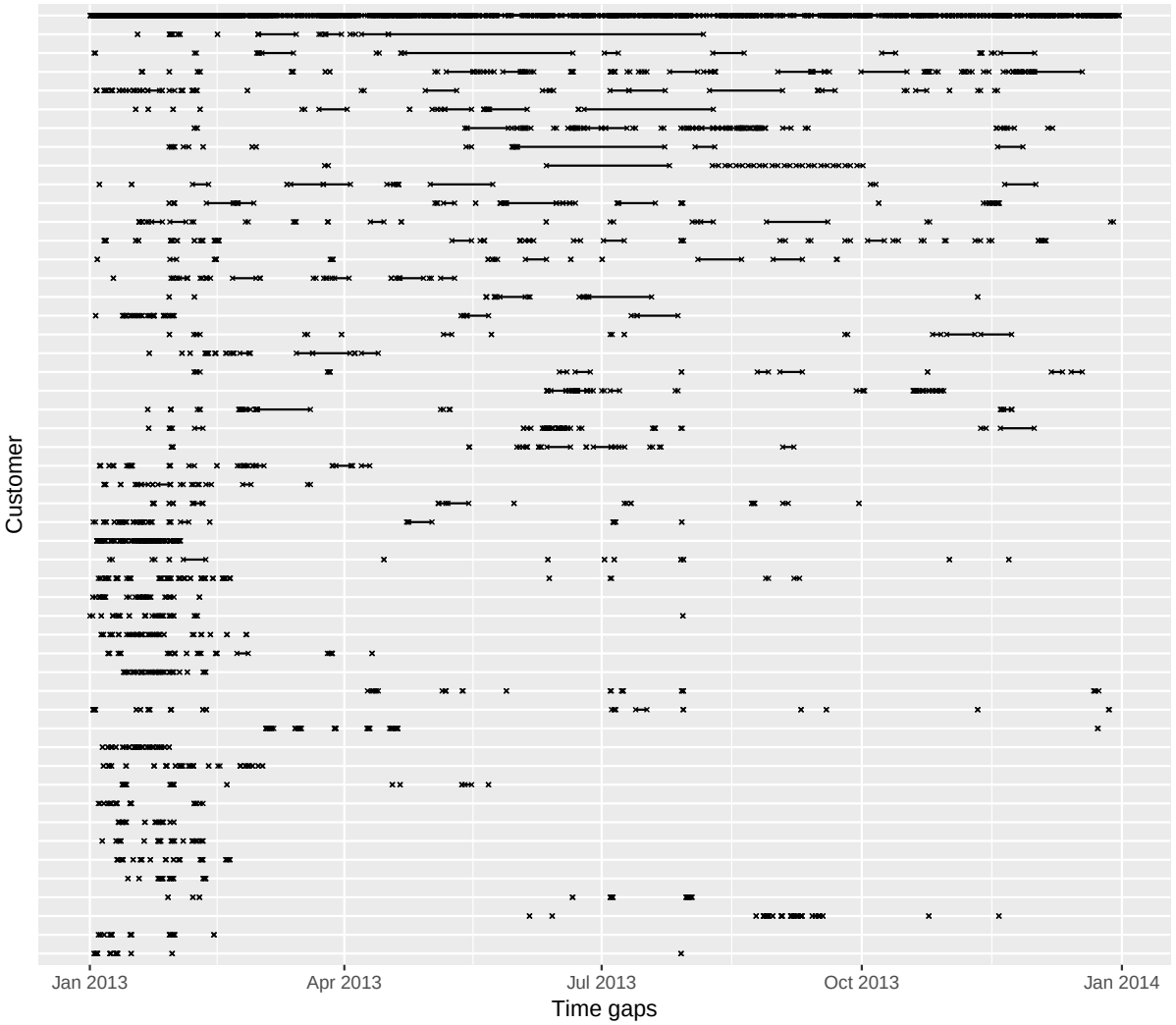

Figure 11: *Time gap plots for the 49 customers with most implicit missing values, and the remaining customers grouped into the one line at top. Each cross represents an observation missing in time and a line between two dots shows continuous missingness over time. Each row corresponds to one customer. Missing values occur at various times, with more in January and February than other months.*

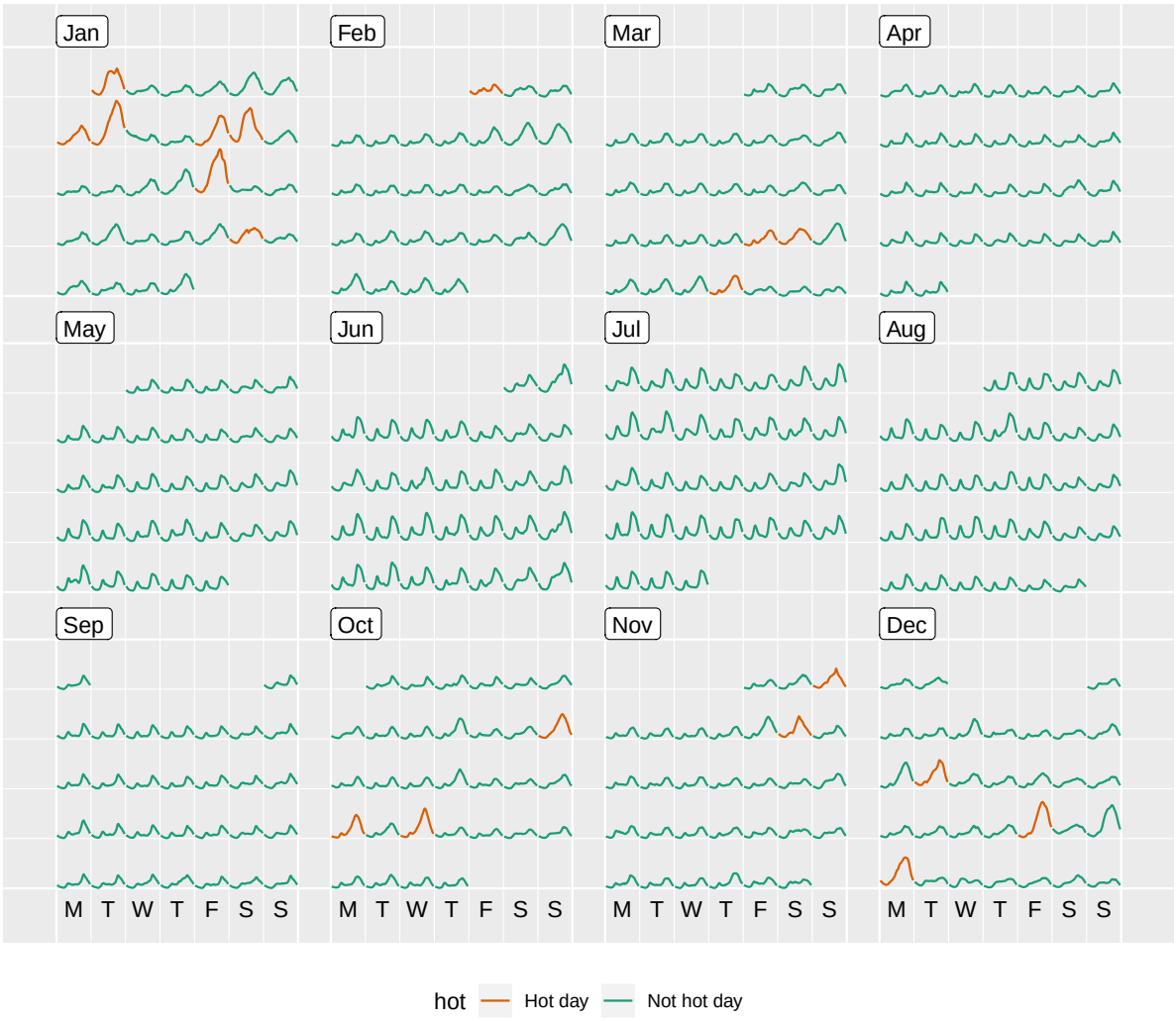

Figure 12: *Half-hourly average electricity use across all customers in the region, organized into calendar format, with color indicating hot days. Energy use of hot days tends to be higher, suggesting air conditioner use. Days in the winter months have a double peak suggesting heater use.*

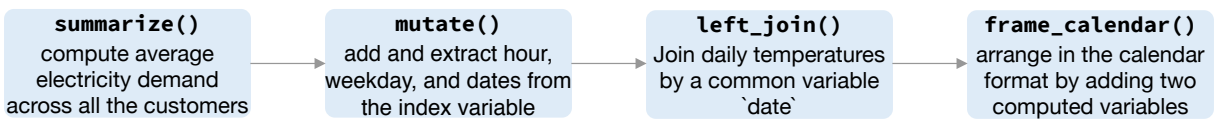

Figure 13: *Flow chart illustrating the pipeline that preprocessed the data for creating Figure 12.*

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.

Acknowledgments

The authors would like to thank Mitchell O'Hara-Wild for many discussions on the software development and Davis Vaughan for contributing ideas on rolling window functions. We also thank Stuart Lee for the feedback on this manuscript. This article was created with knitr (**knitr**) and R Markdown (**rmarkdown**). 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.