tsibble and fable

⚔
with xaringan

Kevin Song

2020-02-19

1) tsibble

2) fable

1) tsibble

```
install.packages("tsibble")
```

> The **`tsibble`** package extends the tidyverse to temporal data.

- Built on top of the tibble, **`tsibble`** is a data- and model-oriented object.
- Comparison to R-based time series objects (`ts`, `zoo` and `xts`)
    - **`tsibble`** preserves time indices as the essential data column and makes heterogeneous data structures possible.
    - Beyond the tibble-like representation, **key** is introduced to uniquely identify observational units over time (index).
    - The tsibble package aims at managing temporal data and getting analysis done in a fluent workflow.

Basic Structure

> To coerce data frame to tsibble, we need to declare **key** and **index**.

- `tsibble()` creates a *tsibble object* and `as_tsibble()` is an S3 method to coerce other objects to a tsibble.
- Vector/Matrix objects can be automated to a tsibble using `as_tsibble()` without any specification.
- For a tibble or data frame, `as_tsibble()` requires to declare the **key** and **index** variables.

Basic Structure

> To coerce data frame to tsibble, we need to declare **key** and **index**.

- In tsibble:

1. **Key** is a set of variables that define subjects measured over time.
    - **identifier**
2. **Index** is a variable with inherent ordering from past to present
    - **time-relevant variable**
3. Each observation should be uniquely identified by **key** and **index**.
4. Other non-identified variables are referred to as *measured* variables.

Contextual semantics: **key** and **index**

- **Key** can be comprised of empty, one, or more variables.
    - For multiple variables, declare `key = c(x1, x2, x3)`.
    - can be created in conjunction with tidy selectors like `starts_with()`
- **Index** supports an extensive range of indices
    - native time classes in R (Date, POSIXct, difftime, etc.)
    - tsibble's new additions (yearweek, yearmonth, and yearquarter)
- **Interval** is computed from index based on corresponding time class.

Class	interval
interger/numeric/ordered	either "unit" or "year"(Y)
yearquarter/yearqtr	"quarter"(Q)
yearmonth/yearmon	"month"(M)
yearweek	"week"(W)
Date / difftime	"day"(D) / W, D, h, m, s
POSIXt, hms / nanotime	h, m, s, us, ms / ns

Ex) nycflights13::weather

- The `weather` data contains the hourly records (temperature, humid and precipitation) over the year of 2013 at three stations (JFK, LGA and EWR).

```
weather <- nycflights13::weather %>%
  select(origin, time_hour, temp, humid, precip)

weather
```

```
## # A tibble: 26,115 x 5
##    origin time_hour            temp humid precip
##    <chr>  <dttm>              <dbl> <dbl>  <dbl>
##  1 EWR    2013-01-01 01:00:00  39.0  59.4      0
##  2 EWR    2013-01-01 02:00:00  39.0  61.6      0
##  3 EWR    2013-01-01 03:00:00  39.0  64.4      0
##  4 EWR    2013-01-01 04:00:00  39.9  62.2      0
##  5 EWR    2013-01-01 05:00:00  39.0  64.4      0
##  6 EWR    2013-01-01 06:00:00  37.9  67.2      0
##  7 EWR    2013-01-01 07:00:00  39.0  64.4      0
##  8 EWR    2013-01-01 08:00:00  39.9  62.2      0
##  9 EWR    2013-01-01 09:00:00  39.9  62.2      0
## 10 EWR    2013-01-01 10:00:00  41     59.6      0
## # ... with 26,105 more rows
```

Ex) nycflights13::weather

- `time_hour` containing the date-times should be declared as **index**
- `origin` should be declared as **key**.
- Other columns can be considered as measured variables.

```
weather_tsbl <- as_tsibble(weather,
                           key = origin,
                           index = time_hour)
weather_tsbl
```

```
## # A tsibble: 26,115 x 5 [1h] <America/New_York>
## # Key:       origin [3]
##    origin time_hour            temp humid precip
##    <chr>  <dttm>              <dbl> <dbl>  <dbl>
##  1 EWR    2013-01-01 01:00:00  39.0  59.4      0
##  2 EWR    2013-01-01 02:00:00  39.0  61.6      0
##  3 EWR    2013-01-01 03:00:00  39.0  64.4      0
##  4 EWR    2013-01-01 04:00:00  39.9  62.2      0
##  5 EWR    2013-01-01 05:00:00  39.0  64.4      0
##  6 EWR    2013-01-01 06:00:00  37.9  67.2      0
##  7 EWR    2013-01-01 07:00:00  39.0  64.4      0
##  8 EWR    2013-01-01 08:00:00  39.9  62.2      0
```

fill_gaps()

> **fill_gaps()** turns implicit missing values into explicit.

- If the observations are made at regular time interval, we could turn these implicit missingness to be explicit simply using `fill_gaps()`.
- `fill_gaps()` also handles filling in **time gaps** by values or functions and respects time zones for date-times.

```
full_weather <- weather_tsbl %>%
  fill_gaps(precip = 0) %>%
#filling gaps in precip with 0
  group_by_key() %>%
  tidyr::fill(temp, humid, .direction = "down")
#replaces NAs with its previous observation for each origin (key)

full_weather
```

```
## # A tsibble: 26,190 x 5 [1h] <America/New_York>
## # Key:       origin [3]
## # Groups:    origin [3]
##    origin time_hour               temp humid precip
##    <chr>  <dttm>                 <dbl> <dbl>  <dbl>
```

fill_gaps()

	origin	time_hour	temp	humid	precip
1	EWR	2013-01-01 01:00:00	39.02	59.37	0
2	EWR	2013-01-01 02:00:00	39.02	61.63	0
3	EWR	2013-01-01 03:00:00	39.02	64.43	0
4	EWR	2013-01-01 04:00:00	39.92	62.21	0
5	EWR	2013-01-01 05:00:00	39.02	64.43	0
6	EWR	2013-01-01 06:00:00	37.94	67.21	0
7	EWR	2013-01-01 07:00:00	39.02	64.43	0
8	EWR	2013-01-01 08:00:00	39.92	62.21	0
9	EWR	2013-01-01 09:00:00	39.92	62.21	0
10	EWR	2013-01-01 10:00:00	41.00	59.65	0
11	EWR	2013-01-01 11:00:00	41.00	57.06	0
12	EWR	2013-01-01 13:00:00	39.20	69.67	0
13	EWR	2013-01-01 14:00:00	39.02	54.68	0
14	EWR	2013-01-01 15:00:00	37.94	57.04	0
15	EWR	2013-01-01 16:00:00	37.04	49.62	0
16	EWR	2013-01-01 17:00:00	35.96	49.83	0

	origin	time_hour	temp	humid	precip
1	EWR	2013-01-01 01:00:00	39.02	59.37	0
2	EWR	2013-01-01 02:00:00	39.02	61.63	0
3	EWR	2013-01-01 03:00:00	39.02	64.43	0
4	EWR	2013-01-01 04:00:00	39.92	62.21	0
5	EWR	2013-01-01 05:00:00	39.02	64.43	0
6	EWR	2013-01-01 06:00:00	37.94	67.21	0
7	EWR	2013-01-01 07:00:00	39.02	64.43	0
8	EWR	2013-01-01 08:00:00	39.92	62.21	0
9	EWR	2013-01-01 09:00:00	39.92	62.21	0
10	EWR	2013-01-01 10:00:00	41.00	59.65	0
11	EWR	2013-01-01 11:00:00	41.00	57.06	0
12	EWR	2013-01-01 12:00:00	41.00	57.06	0
13	EWR	2013-01-01 13:00:00	39.20	69.67	0
14	EWR	2013-01-01 14:00:00	39.02	54.68	0
15	EWR	2013-01-01 15:00:00	37.94	57.04	0
16	EWR	2013-01-01 16:00:00	37.04	49.62	0

index_by() + summarise()

> **`index_by()`** + **`summarise()`** to aggregate over calendar periods

- tidy data representation mostly supports part of a "data pipeline" in time-based context.
- `index_by()` is the counterpart of `group_by()` in temporal context, but it groups the time index only.
- In conjunction with `index_by()`, **`summarise()`** and its scoped variants aggregate variables over calendar periods.
- `index_by()` goes hand in hand with the index functions including `as.Date()`, `yearweek()`, `yearmonth()` and `yearquarter()`, as well as other friends from `lubridate`.

index_by() + summarise()

Example) To compute *average* temperature and *total* precipitation *per month*, we can apply `yearmonth()` to the index variable (referred to as .).

- While collapsing rows (by `summarise()`), `group_by()` and `index_by()` take care of updating the key and index respectively.

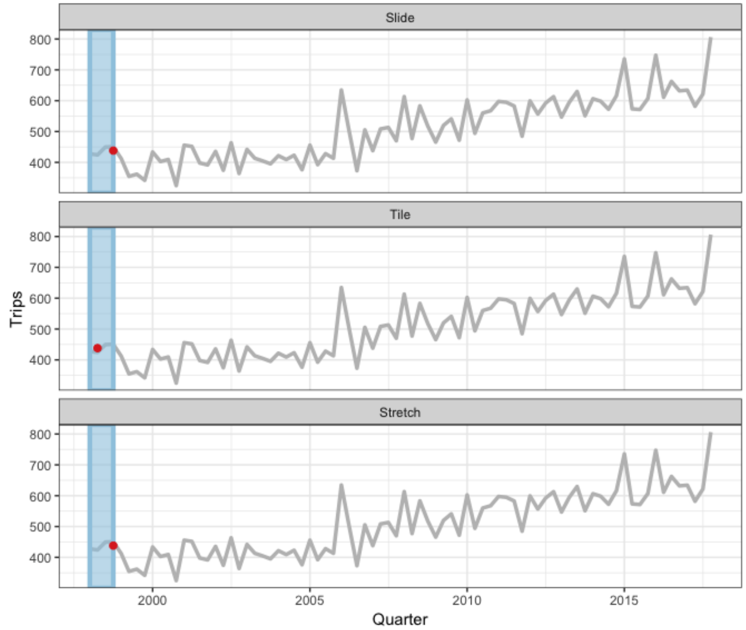
```
full_weather %>%
  group_by_key() %>%
  index_by(year_month = ~ yearmonth(.)) %>%
  # monthly aggregates with new variable name (year_month)
  summarise(
    avg_temp = mean(temp, na.rm = TRUE),
    ttl_precip = sum(precip, na.rm = TRUE)
  )
```

```
## # A tsibble: 36 x 4 [1M]
## # Key:       origin [3]
##    origin year_month avg_temp ttl_precip
##    <chr>       <mth>    <dbl>      <dbl>
##  1 EWR      2013 Jan     35.6       3.53
##  2 EWR      2013 Feb     34.2       3.83
##  3 EWR      2013 Mar     40.1       3
```

Rolling with slide(), tile(), stretch()

- Several functions in tsibble allow for different variations of moving windows using purrr-like syntax:

  - `slide()`: sliding window with overlapping observations.
  - `tile()`: tiling window without overlapping observations.
  - `stretch()`: fixing an initial window and expanding to include more observations.



2) fable

```
install.packages("fable")
```

> **fable** package provides a collection of commonly used univariate and multivariate time series forecasting models

- Forecasing models include exponential smoothing via state space models and automatic ARIMA modelling.
- These models work within the fable framework, which provides the tools to evaluate, visualise, and combine models in a workflow consistent with the tidyverse.

Example

```
p_load(fable, tsibble, tsibbledata, lubridate, dplyr, feast)

aus_retail %>%
  filter(
    State %in% c("New South Wales", "Victoria"),
    Industry == "Department stores"
  ) %>%
  model(
    ets = ETS(box_cox(Turnover, 0.3)),
    arima = ARIMA(log(Turnover)),
    snaive = SNAIVE(Turnover)
  ) %>%
  forecast(h = "2 years") %>%
  autoplot(filter(aus_retail, year(Month) > 2010), level = NULL)
```

> Sorry, I DO NOT know what `ets` and `snaive` are.

- Visit Forecasting: Principles and Practice
  - looks super helpful when dealing with timeseries data

Example