The Journal of Open Source Software

Processing raw mobility data with the checkin package

Michael Kane [1¶], **Owais Gilani** [2], **and Simon Urbanek** [3]

1 Biostatistics Department, School of Public Health, Yale University, New Haven, CT, USA **2** Math Department, Bucknell University, Lewisburg, PA, USA **3** Statistics Department, University of Auckland, Auckland, NZ ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

The analysis of mobility data often begins with the processing of data capturing the time and location of individuals. When preparing these data for analysis, the continuous timestamp data can be particularly difficult to normalize since they require the data scientist to integrate mode assumptions such as, where is a person/device between checkins? Where is a person/device before the first checkin? Should last-observation-carried-forward be used between checkins or between descretization boundaries? For example, we may want to build sequences of checkins for a given device over locations and then count the total number of transitions between locations over all individuals. Problems like these are common in mobility research, they require careful consideration based on the goals of an analysis, and software tools implementing these types of computations will provide benefits in terms of time savings and data integrity. To address these challenges we provide the `checkin` package, which provides a standard set of functions for appropriately descretizing spatio-timestamp data for aggregate analysis for the R programming environment (R Core Team 2022).

Statement of Need

Raw mobility data is often characterized by having a column denoting the device/person identifier, a timestamp, the location, and potentially other features of the checkin. The device/person identifer is often given as a unique identifier of a device or person in possession of a device; the timestamp denotes the time at which a person was at a given location; and the location can be precise location information (such as GPS) but is often aggregated to a discrete location, such as a store, census tract, county etc. While these data are information-rich, to analyze them, especially at the aggregate level (many devices and many locations), requires processing to transform them into a representation amenable to analysis. One such representation is the *mobility graph*, which encodes vertices as discrete locations, directed edges as the aggregate movement between locations, and edge weights capturing the amount of movement (or similar measure) between locations with corresponding directed edge `[@gilani2020]`. More generally, processing steps for these aggregate analyses either descretize continuous timestamp and spatial data and/or aggregate already descretized data.

Basic operations for processing time-stamp are implemented in the core of R and there are a plethora of package that make operations such as reading, comparing, and adding offsets more convenient (see Eddelbuettel (2020), Grolemund and Wickham (2011), etc. for more) there has not been a set of standard functions specifically for processing checking data, which requires the following types of operations:

1. Find the location of person/device at a specified interval, using last-observation-carry forward if specified.
2. Construct a generator for creating sequences of intervals over which data should be processed.

3. A map operation working in conjunction with interval iterators for processing data over spatio-interval data.

The checkin package provides an extensible library for providing these operations in a way that is compatible with the "tidy data" approach described in Wickham (2014), it is compatible with standard dplyr (Wickham et al. 2022) functions, and it uses foreach package (Microsoft and Weston 2022) to provide a parallel-backend for compute-intensive computing. Data sets returned from checkin functions are in the tibble (Grolemund and Wickham 2011) format.

Usage

Consider the checkins data set from the checkin library shown below. The data consists of 3 columns corresponding to the device (id), time, (timestamp), and location identifier (location). There are a total of 1000 unique people/devices rangin in time from 2020-04-19 00:01:13 EST to 2020-05-08 18:58:29 EST.

```
library(checkin)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

data(checkins)
checkins

## # A tibble: 14,609 × 3
##       id timestamp           location
##    <int> <dttm>                 <int>
##  1   442 2020-04-19 17:23:44    36496
##  2   442 2020-04-19 17:23:35    36496
##  3   166 2020-04-19 12:55:44    37461
##  4   476 2020-04-19 06:23:47    33476
##  5   456 2020-04-19 05:39:09    33468
##  6   458 2020-04-19 10:04:20    36500
##  7   458 2020-04-19 15:15:25    36876
##  8   651 2020-04-19 13:31:19    37391
##  9   652 2020-04-19 05:27:54    37469
## 10   653 2020-04-19 05:28:59    37389
## # … with 14,599 more rows
## # □ Use `print(n = ...)` to see more rows
```

Now suppose we would like to examine the checkins of indvidual/device number 335. We will extract all rows with id 335 and then specify an interval starting at the beginning of the hour of the first checkin to one hour later using the checkins_in_interval() function. The code and output are below and there are two things to note. First, the first row location is NA. This is because the individual/device does not appear before 2020-04-19 00:41:46 and a location cannot be determined. Second the original data set did not include an entry for id 335 at 2020-04-19 00:59:59. This value was carried forward from the previous location.

```
library(lubridate)

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

x <- checkins %>%
  filter(id == 335) %>%
  arrange(timestamp)

start <- x$timestamp[1]
minute(start) <- 0
second(start) <- 0

end <- start + hours(1) - seconds(1)
checkins_in_interval(x, "timestamp", start, end)

## # A tibble: 3 × 3
##       id timestamp           location
##    <int> <dttm>                 <int>
## 1     NA 2020-04-19 00:00:00       NA
## 2    335 2020-04-19 00:41:46    32576
## 3    335 2020-04-19 00:59:59    32576
```

Finally, suppose we would like to get the locations of individuals/devices at the beginning of a time interval, the location at the end of the interval, and the total amount of time the individual/devices was checked into the beginning location. This operation would be performed in two steps. First, a function, `from_to()` is constructed, which takes the rows corresponding to a single `id` for a given interval. This function finds, the initial location (`from`), the end location(`to`), the timestamp at the *beginning* of the interval, and the duration of the initial location. In the second step, we group the checkin data by `id` and pass the result to the `map_hourly_interval_dfr()` function, which applies to `from_to()` function to each `id` over each hourly interval. Other intervals are included in the package and the documentation includes a information on how to construct similar functions over custom intervals.

```
from_to <- function(it) {
  it$duration <- c(diff(it$timestamp), 0)
  units(it$duration) <- "secs"
  it$duration <- as.numeric(it$duration)
  from_duration <- sum(it$duration[it$location == it$location[1]])
  tibble(from=it$location[1],
         to = it$location[nrow(it)],
         timestamp = it$timestamp[1],
         from_duration = from_duration)
}

 checkins |>
   head(100) |>
   group_by(id) |>
   map_hourly_interval_dfr(from_to, time = "timestamp")

## # A tibble: 119 × 5
##     id     from     to timestamp           from_duration
##    <chr> <int> <int> <dttm>                      <dbl>
##  1 166   37461 37461 2020-04-19 13:00:00          3599
##  2 286   33165 33165 2020-04-19 18:00:00          3599
##  3 313   33149 33149 2020-04-19 19:00:00          3599
##  4 381   33359 33359 2020-04-19 16:00:00          3599
##  5 381   33359 33359 2020-04-19 17:00:00          3599
##  6 442   36496 36496 2020-04-19 18:00:00          3599
##  7 456   33468 33468 2020-04-19 06:00:00          3599
##  8 457   33130 33130 2020-04-19 13:00:00          3599
##  9 457   33130 33130 2020-04-19 14:00:00          3599
## 10 457   33130 33130 2020-04-19 15:00:00          3599
## # … with 109 more rows
## # □ Use `print(n = ...)` to see more rows
```

Acknowledgements

This work was partially supported by the National Science Foundation (NSF) Grant Human Networks and Data Science - Infrastructure (HNDS-I), award numbers 2024335 and 2024233.

References

Eddelbuettel, Dirk. 2020. *Anytime: Anything to 'POSIXct' or 'Date' Converter*. https://CRAN.R-project.org/package=anytime.

Grolemund, Garrett, and Hadley Wickham. 2011. "Dates and Times Made Easy with lubridate." *Journal of Statistical Software* 40 (3): 1–25. https://www.jstatsoft.org/v40/i03/.

Microsoft, and Steve Weston. 2022. *Foreach: Provides Foreach Looping Construct*. https://CRAN.R-project.org/package=foreach.

R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10): 1–23. https://doi.org/10.18637/jss.v059.i10.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2022. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.