sugarglider: Create glyphmaps of spatio-temporal data

by Maliny Po and Dianne Cook

Abstract (An abstract of less than 150 words.)

1 Introduction

```
Note: use similar terminology as cubble & glyph-maps. Add a quick start (quick guide on how to use the sugarglider
```

2 Literature Review

Glyph Maps

```
(Desperately need more resources)
```

- Glyph maps are a specific type of multivariate glyph plots where each spatial location is represented by a glyph that encapsulates data measured over time at that location. As detailed in Hadley Wickham's paper, glyph maps are particularly effective for uncovering both local and global structures, emphasizing temporal relationships within the data. These maps utilize small glyphs or icons to represent multiple data values at each location, extending the concept of glyphs which are traditionally used to display multivariate data.
- Challenges with Faceted Maps and Spatio-Temporal Animations: While faceted maps and spatio-temporal animations are useful for highlighting spatial patterns, they often fall short in adequately showcasing temporal trends. To overcome this, a transformation technique is employed which utilizes principles of linear algebra to convert temporal coordinates (minor coordinates) into spatial coordinates (major coordinates). This transformation is implemented in packages such as GGally and cubble, facilitating a more integrated approach to spatio-temporal data visualization.
- The R package cubble introduces an innovative cubble class designed to efficiently organize spatial and temporal variables. This dual structure allows for separate or combined manipulation of these variables while maintaining synchronization. A spatial cubble object is constructed from distinct spatial and temporal tables through the function make_cubble(), requiring the specification of three attributes: key, index, and coords. This functionality not only simplifies the data wrangling process but also enhances the analytical capabilities when dealing with complex datasets.

Extending ggplot2 with ggproto

```
 Diversify your resources a bit :((
```

- Elegant Graphics for Data Analysis: The architecture of ggplot2 is fundamentally based on the ggproto system of object-oriented programming. Initially, ggplot2 utilized the proto system, developed by Grothendieck, Kates, and Petzoldt in 2016, for object-oriented tasks. This system, described in detail in the Proto package documentation, outlines that proto is an S3 subclass of the R environment class, implying single inheritance and mutable state characteristic of all environments. Proto objects are created and modified using the proto function which sets the parent environment, evaluates expressions, and handles lazy evaluation of arguments.

However, as the need for an official extension mechanism in ggplot2 grew, the limitations of the proto system became apparent, leading to the adoption of ggproto. This transition is well-documented in Hadley Wickham's book, ggplot2: Elegant Graphics for Data Analysis, which also introduces how to utilize ggproto objects to extend ggplot2 functionalities.

The creation of a new ggproto object is facilitated by the ggproto() function, which requires the name of the new class and an existing ggproto object from which it will inherit. For instance, to introduce a new statistical transformation, one might create a ggproto that inherits from `Stat` and `Geom`. However, merely creating a ggproto object does not make it accessible or useful to the end user.

(Example from ggplot2-book.org)

```
NewObject <- ggproto(
  `_class` = NULL,
  `_inherits` = NULL
)
```

To bridge this gap, the creation of a layer function is necessary. An example is the `new_stat()` function, which follows a consistent format: setting defaults in the function arguments, and calling layer(), which handles the distribution of these arguments into either geom parameters, stat parameters, or aesthetics. This function exemplifies the methodology to create functional and user-accessible components in ggplot2.

(Example from ggplot2-book.org)

```
new_stat <- function(mapping = NULL, data = NULL,
                       geom = "geometry", position = "identity",
                       na.rm = FALSE, show.legend = NA,
                       inherit.aes = TRUE, ...) {
  layer(
    stat = NewStat,
    data = data,
    mapping = mapping,
    geom = geom,
    position = position,
    show.legend = show.legend,
    inherit.aes = inherit.aes,
    params = list(na.rm = na.rm, ...)
  )
}
```

While developing ggplot2 extensions, it may seem intuitive to encapsulate extensions as new geoms, as they are frequently used by users to add layers to a plot. However, the diversity in ggplot2's capabilities often stems more from the variety in statistical transformations (stats) than merely geometric objects (geoms), suggesting a nuanced approach in designing extensions that effectively enhance the plotting system.

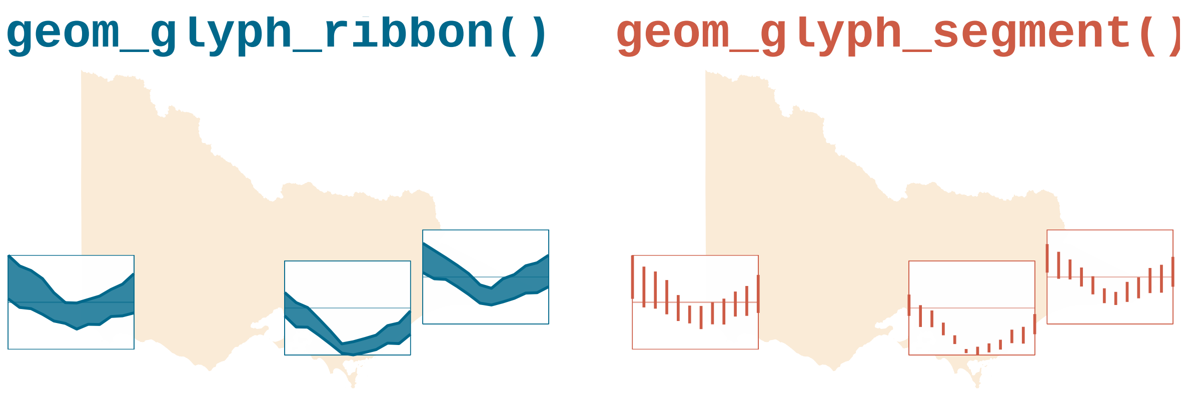
3 Software

```
Check out:  https://journal.r-project.org/articles/RJ-2023-013
Note: collect all the technical part in one section
```

`sugarglider` provides ggplot2 extensions to create glyph maps that visualize seasonal aspects of spatio-temporal data with geom_glyph_ribbon() and geom_glyph_segment().

Designed to visualize seasonal aspects of spatio-temporal data, sugarglider has the capability to display glyphs with ribbon or segment geometry based on the combination of x_major and y_major. Figure **??** highlights the package's versatility. The code for these examples is provided below and can be summarised as consisting of the following components. First the glyph plot is created by adding either ribbon or segment geom from ggplot2. For each x_minor value, geom_glyph_ribbon() displays a y interval defined by ymin_minor and ymax_minor. Meanwhile, geom_glyph_segment() draw a straight line between y_minor and yend_minor with respect to x_minor. An optional global_rescale argument allow users to control whether rescalling is applied globally across all glyphs or individually for each glyph.

`add_ref_line`, `add_ref_box`, and `add_geom_legend`



Furthermore, sugarglider allows various features to be customised: - **Scaling of minor values** with-in a grid cell along the x and y dimension. - **Rescale** of the value in ploted in each glyph. Determines whether rescaling is applied globally across all glyphs or individually for each glyph. - **Width** and **height** of the glyphs.

The following section will discuss these features and provide examples of their use.

Data structure

The first step to creating a glyph plot with sugarglider is to ensure that the data are in supported format. There are 2 data structure to consider Zhang et al. (2024), one of which is compatible with sugarglider. sugarglider exspect data in a long format with temporal and spatial elements.

One dataset provided in the package is the `aus_temp` dataset which derives from The National Oceanic and Atmospheric Administration (NOAA) provides comprehensive weather data from numerous stations across Australia. The aus_temp dataset includes key climate variables, such as precipitation and temperature, recorded at 29 different weather stations throughout 2020. It has spatial elements (longitude, latitude), temporal elements (month) and range of temperature that define the width of ribbon and segment plot.

```
glimpse(aus_temp)

#> Rows: 348
#> Columns: 7
#> $ id    <chr> "ASN00001020", "ASN00001020", "ASN00001020", "ASN00001020", "ASN~
#> $ long  <dbl> 126.3867, 126.3867, 126.3867, 126.3867, 126.3867, 126.3867, 126.~
#> $ lat   <dbl> -14.0900, -14.0900, -14.0900, -14.0900, -14.0900, -14.0900, -14.~
#> $ month <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9~
#> $ tmin  <dbl> 253.4516, 248.6786, 253.6129, 244.0357, 220.4138, 202.3667, 153.~
#> $ tmax  <dbl> 319.0000, 322.6071, 333.1935, 340.9310, 331.9333, 310.9000, 291.~
#> $ prcp  <dbl> 163.87096774, 162.74074074, 42.00000000, 21.57142857, 0.00000000~
```

In many cases, analysts may initially receive station data containing geographic location information, recorded variables and their recording periods. They can then query the temporal variables using the stations of interest to obtain the relevant temporal data. Alternatively, analyses may begin as purely spatial or temporal, and analysts may obtain additional temporal or spatial data to expand the result to spatio-temporal. In this case, users can compose cubble objects using functions from cubble packages such as make_cubble() and pass the new data to sugarglider.

Rescale

sugarglider rescale minor axies (data that are used to plot the individual glyphs) before plotting the glyphs. The purpose of rescaling the value to prepare the data for the linear transformation to map temporal data into a spatial one, which will be explored in the next section. The rescale is controlled by 2 parameters: `x_scale` and `y_scale`. `x_scale` scales each set of minor values within a grid cell along the x-dimension. `y_scale` scales each set of minor values within a grid cell along the y-dimension. The default rescale is 'identify' which adjust minor axes to to fit within an interval of [-1,1] using the following equation.

Users can also change the rescale function by replacing the default value for `x_scale` and `y_scale` with their own function. In the example code below, we have a custom rescale function that transform value into an interval of [0,1].

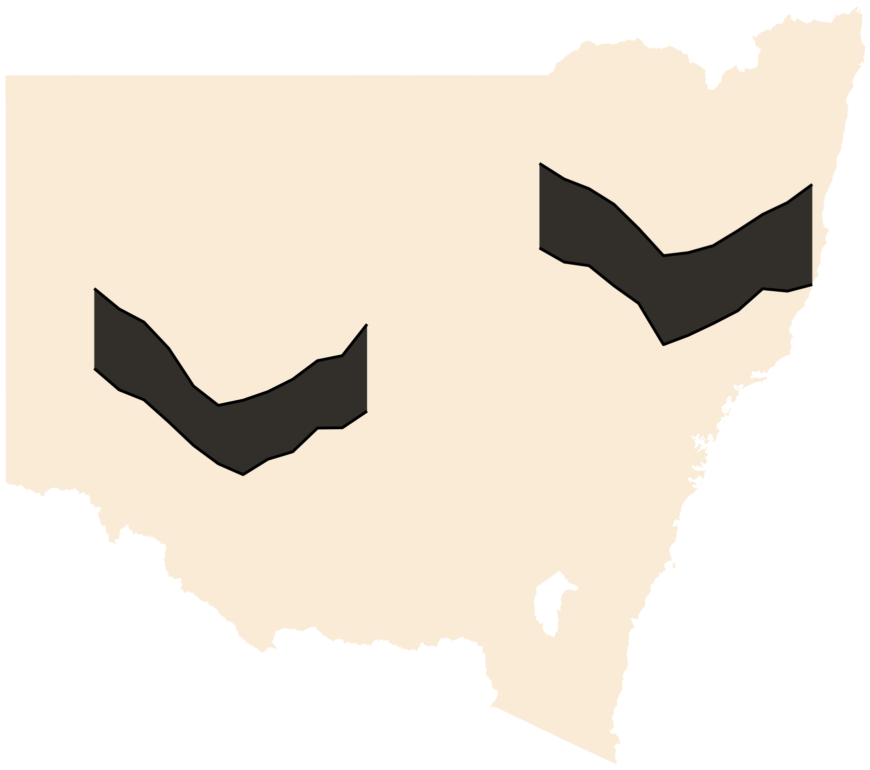


Figure 1: Default rescale. Additional codes are needed for plotting the base map

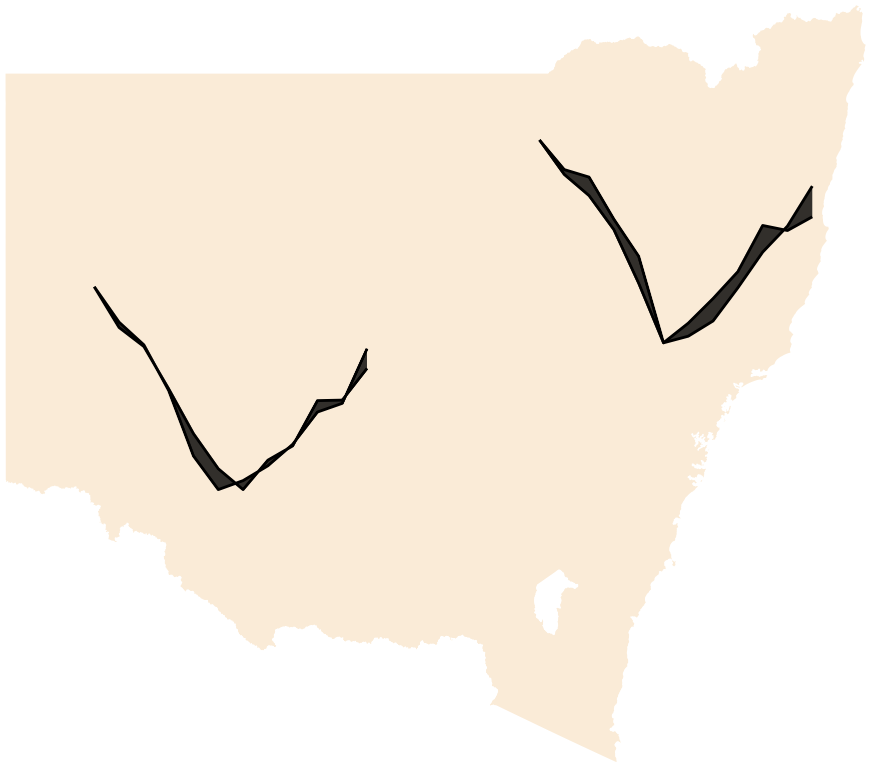
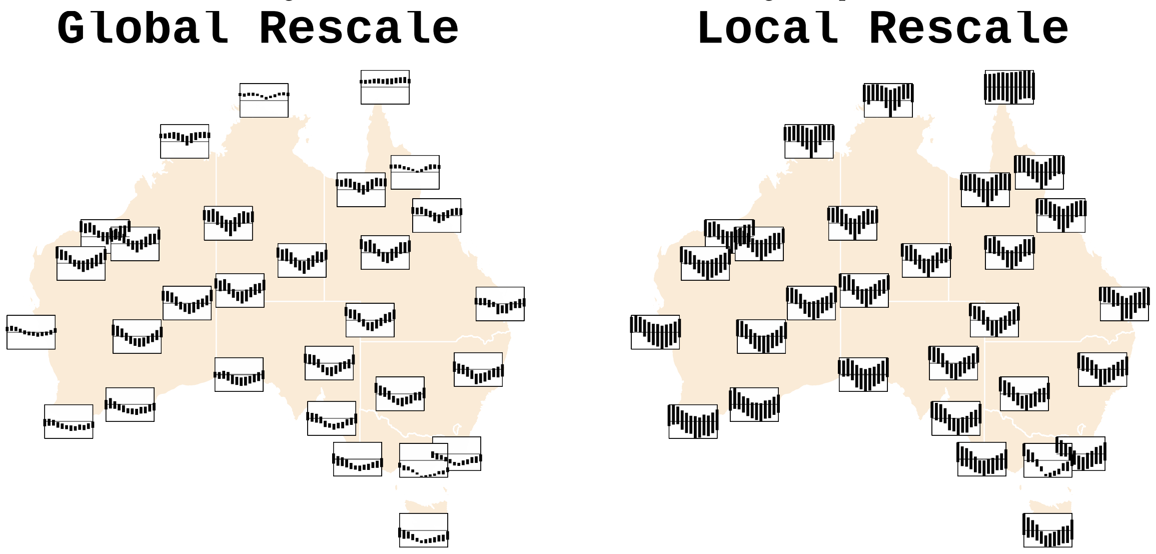


Figure 2: Custom rescale. Additional codes are needed for plotting the base map

With the new rescale functions, we can observe that the ribbon are alot thinner than the previous example. To understand how rescale impact the mapping of temporal data into glyphs, we need to look at the how rescale is applied for both `geom_glyph_ribbon` and `geom_glyph_segemnt` and the spatial temporal transformation for each of these plot type in the following section.

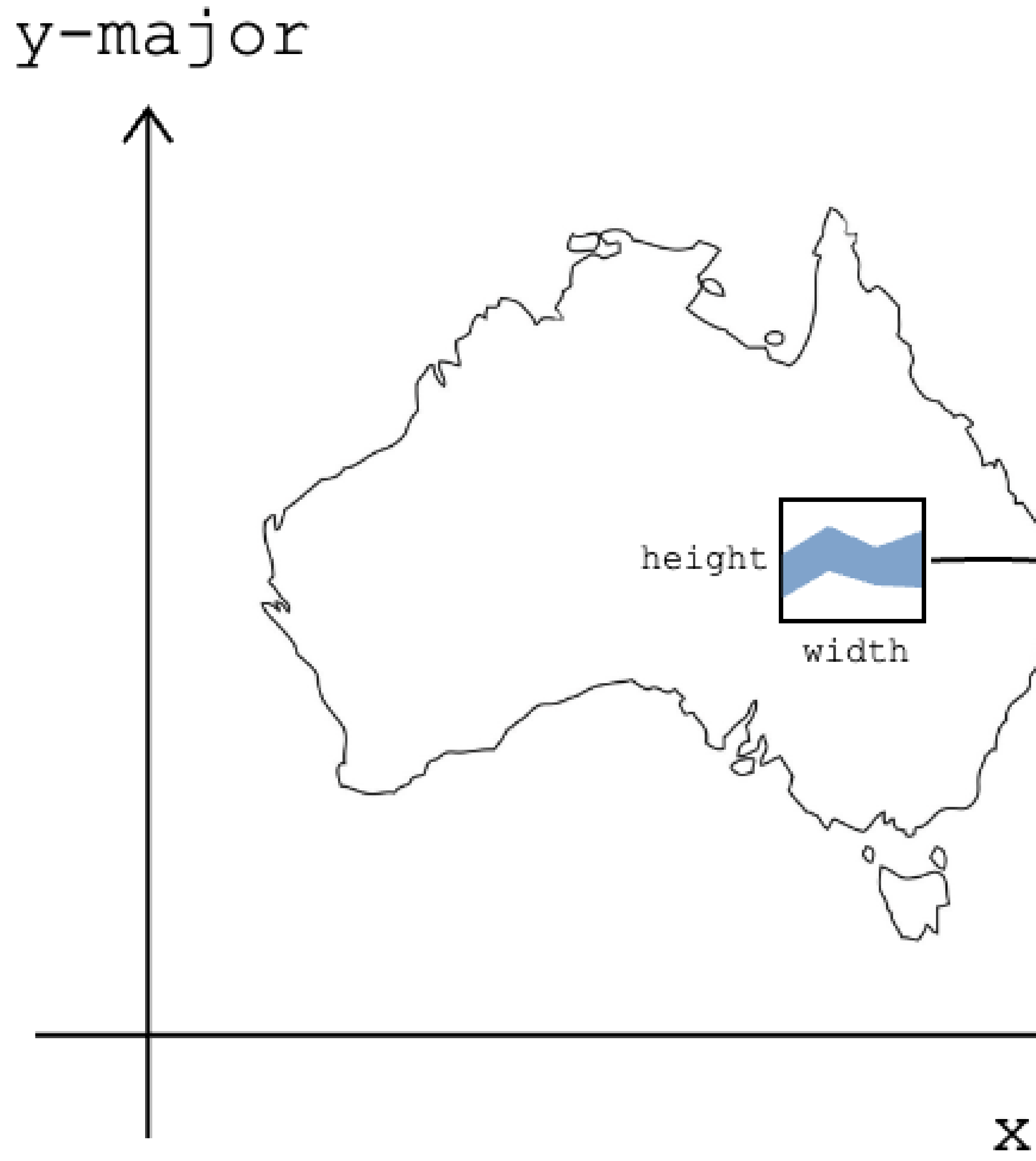
sugarglider also allow user to choose whether rescaling is applied globally across all glyphs or individually for each glyph. This is controlled by an option called `global_rescale`. The default for this parameter is `TRUE`. User can specify for a local rescale by setting `global_rescale` to be `FALSE`. The differences between local and global rescale can be observed in the following example:



sugarglider apply rescaling in multiple steps. First the (. . .) For `geom_glyph_ribbon` the rescale is done separately for ymin_minor and ymax_minor.

(. . . . . . .)

Spatial-temporal transformation



The construction of glyph map (as described in Wickham et al. (2012)) involve a linear mix of two structure component of the data: spatial location and data value. The spatial location is the major axes are latitude (y_{major}) and longitude (x_{minor}), and the minor axes are time (x_minor) and some measurement ($ymax_{minor}$ and $ymin_{minor}$). For `geom_segment_glyph` the aesthetic would be y_{minor} and $yend_{minor}$. Once the minor axes are rescaled to [-1,1], the final coordinates (x,$ymin$, $ymax$) for ribbon glyph are the linear combination, given by:

$x = x_{major} + \frac{width}{2}.x_{minor}$ $ymin = y_{major} + \frac{height}{2}.ymin_{minor}$ $ymax = y_{major} + \frac{height}{2}.ymax_{minor}$

Similarly, the coordinates for segment glyph are given by:

$x = x_{major} + \frac{width}{2}.x_{minor}$ $y = y_{major} + \frac{height}{2}.y_{minor}$ $yend = y_{major} + \frac{height}{2}.yend_{minor}$

(...)

Aesthetics

sugarglider provides the same aesthetics for geom_glyph_ribbon() and geom_glyph_segment() as those available in geom_ribbon() or geom_segment(), while also introducing additional unique options.

Parameters

Interactivity

Examples

4 Application

Five examples are selected to demonstrate various features of the sugarglider package: (1) Creating a ribbon glyph map to visualize annual fluctuations in minimum and maximum daily patronage for each train station, revealing seasonal trends. (2) Using glyph segments to compare patronage on typical weekdays versus weekends at different stations, enabling insights for optimizing service schedules. (3) Utilizing glyph ribbons to represent variations in patronage during distinct peak periods—AM peak, interpeak, PM peak, and late PM hours—across stations. (4) Displaying differences in patronage across transportation modes (Metro, VLine, or both) using glyph segments, identifying capacity imbalances. Lastly, (5) Employing glyph ribbons to compare public and school holiday patronage against regular days, aiding in scheduling and resource planning.

Yearly Patronage Changes by Station

Weekday vs. Weekend Patronage

Patronage During Different Peak Times

Patronage Variations by Transportation Mode

Public and School Holiday Patronage vs. Regular Days

5 Discussion

6 Acknowledgements

References

Wickham, Hadley, Heike Hofmann, Charlotte Wickham, and Dianne Cook. 2012. "Glyph-Maps for Visually Exploring Temporal Patterns in Climate Data and Models." *Environmetrics* 23 (5): 382–93.

Zhang, H. Sherry, Dianne Cook, Ursula Laa, Nicolas Langrené, and Patricia Menéndez. 2024. "Cubble: An r Package for Organizing and Wrangling Multivariate Spatio-Temporal Data." *Journal of Statistical Software* 110 (7): 1–27. https://doi.org/10.18637/jss.v110.i07.

Maliny Po
Monash University
Department of Econometrics and Business Statistics

Melbourne, Australia
ORCiD: 0009-0008-4686-6631
`malinypo12@gmail.com`

Dianne Cook
Monash University
Department of Econometrics and Business Statistics
Melbourne, Australia
ORCiD: 0000-0002-3813-7155
`dicook@monash.edu`