The State of Trajectory Visualization in Notebook Environments

GI_Forum 2022, Issue 2
Page: 73 - 91
Short Paper
Corresponding Author:
anita.graser@ait.ac.at
DOI: 10.1553/giscience2022_02_s73

Anita Graser
AIT Austrian Institute of Technology, Austria

Abstract

Gaining insights from trajectory datasets is a challenging task that requires suitable visual data representations. There is a considerable gap between the state-of-the-art cartographic techniques presented in the literature and currently available spatial data science toolboxes. This review paper presents the current state of geospatial visualization tools for trajectory data, focusing on the Python and Jupyter notebooks ecosystem. The shortcomings identified provide pointers for further scientific software development, as well as a reference for data scientists in choosing the best-fitting tool for a specific job.

Keywords:

trajectories, movement data analysis, visual analysis, exploratory data analysis

1 Introduction

GIScience sometimes seems stuck in a continuous search for more theory to 'justify its science credentials' (Gahegan, 2020). By comparison, scientific software development and data provisioning 'have historically been undervalued' (Rey, 2021). As a result, many analysis and visualization approaches developed over the years are not accessible through readily available (proprietary, or free and open-source) software. Consequently, the potential impact of this research on the wider data science community in academia and industry is limited.

Movement data is used in many data science domains, from health to logistics and beyond. However, movement data is rarely collected in lab settings. Many datasets, such as floating car data, call detail records, as well as sports, or ship- and plane-tracking data, are created for purposes other than the scientific analyses that they are later used for. Therefore, data quality – as in fitness for use in analyses – is rarely ideal. Understanding data quality is essential for choosing suitable analysis methods and interpreting their results. However, gaining a proper understanding of a dataset's potential and limitations is a time-consuming task. Graphical data exploration tools in particular are needed to support analysts (Graser & Dragaschnig, 2020).

Multiple recent survey papers provide overviews of visualization techniques for movement data (Chen et al., 2015; Andrienko et al., 2017; He et al., 2019). These visualization techniques are not limited to spatial visualizations. For example, He et al. (2019) present radial icon visualization, which aims to visualize multivariable attributes simultaneously. However, for the purpose of this paper, we will focus on spatial visualizations of trajectories.

To perform a comprehensive data quality assessment, it is important to look at the raw data and not just the processed and aggregated derivatives, which may be influenced by the processing and aggregation steps (Graser, 2021). This paper therefore focuses on visualizations of individual raw trajectories rather than on the numerous different aggregated visualizations.

Many approaches presented in the literature (Chen et al., 2015; Andrienko et al., 2017; He et al., 2019) require access to dedicated visual analytics software, fully-fledged GIS software and potential spatiotemporal extensions, or (often unpublished) research prototypes. However, these tools are hard to integrate into everyday data scientists' workflows, which are more likely to use general-purpose business intelligence (BI) tools, integrated development environments (IDEs), or notebook environments such as Jupyter or RStudio. In this paper, we specifically review the current state of the art in Python libraries, since Python is the scripting language of choice for many scientists and data analysts, in geographic data science in particular and in data science in general. The paper aims to provide readers (with or without a GIScience or cartography background) with the necessary information to make informed choices when faced with the current fragmented spatial data visualization ecosystem.

The remainder of this paper is structured as follows: Section 2 describes the visual variables that are commonly used to visualize trajectory information. Section 3 analyses the current state of trajectory analysis and spatial visualization libraries that are used to provide trajectory visualizations. Finally, Section 4 draws conclusions and aims to put them in a broader context.

2 Spatial trajectory visualization

This section provides a brief overview of visual variables that can be used to visualize trajectory data. While visual variables are a basic topic in cartography and visual/exploratory data science curricula, it is worth revisiting these theoretical fundamentals briefly before reviewing the implementations that are actually available.

To understand trajectories, analysts look at spatial, temporal and potential additional thematic dimensions. In addition to the pure location or geographic context of the movement, key information of interest relates to direction and speed of movement. This information has to be encoded using visual variables, such as shape, size, colour (hue, value, saturation), patterns or orientation.

Raw movement data usually comes in the form of discrete timestamped location records for which reporting intervals (regular or irregular) can vary widely, from milliseconds to years. Most unspecialized spatial visualization tools therefore render raw movement data as points. Depending on the reporting interval, it can be difficult to determine how these points are connected – that is, which points are consecutive locations that belong to the same moving object. To make the data easier to understand, movement data visualizations usually transform the raw data from points into lines between consecutive points and render those.

Lines alone are insufficient to communicate movement direction. Common approaches to visualize the movement direction therefore include:

1. *Markers at start and end locations*, which may be colour-coded, with one colour signifying the trajectory start or origin and the other signifying the end or destination.
2. *Arrow-shaped markers*, which may be placed at the end or along the line to communicate directionality.
3. *Colour or size gradients (tapering)*, which can be applied to point markers or line segments. For example, older positions (closer to the start) may be symbolized using less saturated colours, smaller markers or thinner lines.
4. *Animations*, which are another intuitive way to visualize directionality. Animation options range from a single animated marker that moves along the trajectory path with or without leaving a visible trace, to animated line patterns.
5. *3D*. When time is used as the third dimension in three-dimensional visualizations, such as space-time cubes, directionality is given by the incline of the line.

The visual variables that can be used to communicate speed overlap with those used for direction. This can lead to conflicts. For example, if line colour is already used to show direction, another variable needs to be found to display speed on the same map. Common approaches to visualize speed include:

1. *Colour gradients*, especially traffic-light colour gradients. These are an intuitive and therefore popular choice, even though they are less than ideal for people with colour blindness.
2. *(Arrow) marker-based line decorations* using different spacing between arrow heads or varying line width. However, both line width and arrow spacing are potentially ambiguous and less intuitive. Is a wide line faster than a narrow one? Or are lines with densely spaced arrow markers faster than those with larger gaps?
3. *Density maps*. If trajectories are sampled with regular time intervals, density or heat maps can be used to communicate speed, since trajectory point density will be higher in regions of slower movement. However, when sampling intervals are irregular, heat maps can be misleading and therefore cannot be recommended without reservation.

3 Survey of trajectory visualization tools

This section presents a survey of Python trajectory analysis libraries that provide trajectory visualizations. Table 1 lists these trajectory analysis libraries and the visualization libraries they depend on. The three most popular libraries (according to GitHub's star count) are: MovingPandas, scikit-mobility and TransBigData. While Traja and Trackintel rely on Matplotlib and therefore only support static visualizations, all other libraries provide interactive trajectory maps using one of three options: Folium, GeoViews or Kepler.gl.

Table 1: Overview of open-source Python movement analysis libraries and associated visualization libraries, ordered chronologically from the earliest to the latest GitHub publication date.

Library name	**References**	**Source code repository** (number of stars, July 2022)	**Visualization library**
PyMove	Oliveira (2019)	https://github.com/InsightLab/PyMove (65 stars)	Folium and Matplotlib
MovingPandas	Graser (2019)	https://github.com/anitagraser/movingpandas (834 stars)	GeoViews and Matplotlib
Traja	Shenk et al. (2021)	https://github.com/traja-team/traja (60 stars)	Matplotlib
Trackintel		https://github.com/mie-lab/trackintel (79 stars)	Matplotlib
scikit-mobility	Pappalardo et al. (2019)	https://github.com/scikit-mobility/scikit-mobility (522 stars)	Folium
PTRAIL	Haidri et al. (2021)	https://github.com/YakshHaranwala/PTRAIL (10 stars)	Folium
TransBigData	Yu & Yuan (2022)	https://github.com/ni1o1/transbigdata (182 stars)	Kepler.gl and Matplotlib

3.1 Trajectory analysis libraries

The following figures show plots created by the visualization functions of the trajectory analysis libraries in Table 1. PyMove and PTRAIL are not included due to errors when calling the visualization functions. (These errors are documented in the complete Jupyter notebook provided on Zenodo.[1]) The data used in this notebook is the open science 'Heterogeneous Integrated Dataset for Maritime Intelligence, Surveillance, and Reconnaissance' (Ray et al., 2018). A subset of this dataset (10,000 rows) was loaded as a Pandas DataFrame, which is used as input in all subsequent examples. As a first step, we analyse the decisions made by the individual library development teams and the resulting visualizations. In the second step, we compare these results.

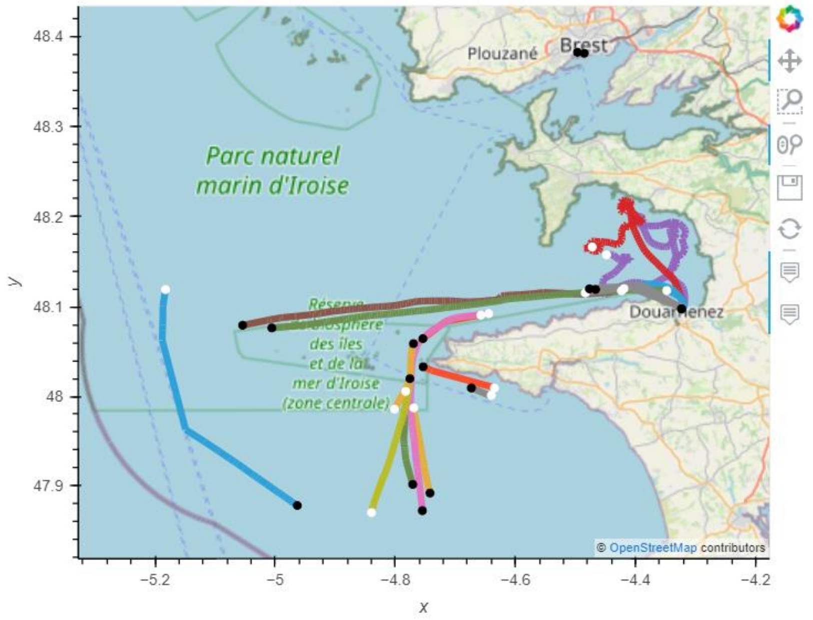
MovingPandas provides static plots using Matplotlib, and interactive maps using HoloViews GeoViews (based on Bokeh). The default interactive plot of a set of trajectories (represented by a TrajectoryCollection object) draws each trajectory in a different colour. As shown in Figure 1, background map tiles, as well as markers for trajectory start and end locations, can be readily added and customized. Pop-ups attached to lines and point markers provide additional information, such as the object ID source mmsi.

Instead of using the colour to indicate object ID, other DataFrame column names (or the keyword 'speed') can be specified to colour the trajectory segments, as shown in Figure 2, where colour indicates the speed of movement.

[1] https://doi.org/10.5281/zenodo.7185322

```
In [7]:  tc = mpd.TrajectoryCollection(df, 'sourcemmsi', x='lon', y='lat', t='t', min_length=1000)
         tc = mpd.MinTimeDeltaGeneralizer(tc).generalize(tolerance=timedelta(minutes=1))
         hvplot_defaults = {'tiles':'OSM', 'frame_height':400, 'frame_width':500, 'line_width':5.0}
         basic_plot = (tc.hvplot(hover_cols=['sourcemmsi'], **hvplot_defaults) *
                       tc.get_start_locations().hvplot(geo=True, c='white') *
                       tc.get_end_locations().hvplot(geo=True, c='black'))
         basic_plot
```

Out[7]:



48.4
48.3
48.2
48.1
48
47.9
y
-5.2
-5
-4.8
-4.6
-4.4
-4.2
x
Plouzané
Brest
Parc naturel marin d'Iroise
Réserve de biosphère des îles et de la mer d'Iroise (zone centrale)
Douarnenez
© OpenStreetMap contributors

Figure 1: MovingPandas / GeoViews plot of a TrajectoryCollection, enhanced with white markers at trajectory start locations and black markers at end locations, superimposed on OpenStreetMap tiles. As a pre-processing step, trajectories are generalized to improve rendering speed. On mouse over, the interactive plot shows attribute values which can be customized using the hover_cols keyword.

```
In [8]: speed_plot = tc.hvplot(title=f'Speed (m/s)', c='speed', cmap='Viridis_r', colorbar=True, **hvplot_
        speed_plot
```

Out[8]:

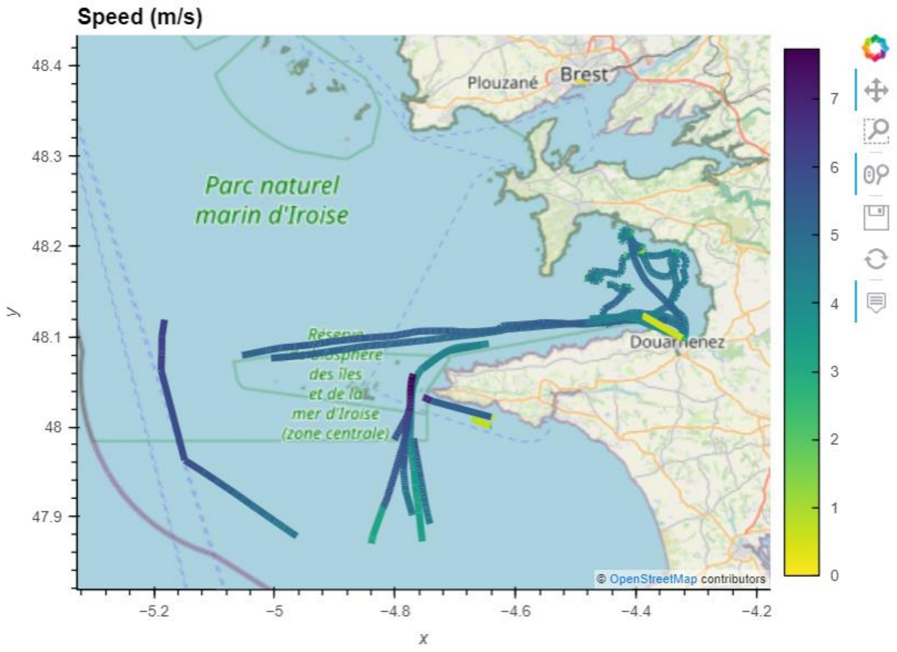

Speed (m/s)
Parc naturel marin d'Iroise
Plouzané
Brest
Douarnenez
Réserve de biosphère des îles et de la mer d'Iroise (zone centrale)
© OpenStreetMap contributors
48.4
48.3
48.2
48.1
48
47.9
y
-5.2
-5
-4.8
-4.6
-4.4
-4.2
x
7
6
5
4
3
2
1
0

Figure 2: MovingPandas / GeoViews plot of a TrajectoryCollection where line segments are coloured by movement speed (values in m/s). If there is no speed column in the Trajectory DataFrame, the speed is computed automatically based on linear interpolation between consecutive records

scikit-mobility provides interactive plots using Folium. The default plot for a TrajDataFrame draws each trajectory in a different colour and automatically puts green and red markers at the start and end locations respectively, as shown in Figure 3. Pop-ups attached to the start and end location markers provide timestamp and coordinate information.

To enhance rendering performance, the plot function by default does not plot all trajectories, and it generalizes the trajectories rendered. These preprocessing steps are communicated to the user in the form of UserWarnings: 'Only the trajectories of the first 10 users will be plotted. Use the argument \`max_users\` to specify the desired number of users, or filter the TrajDataFrame' and 'If necessary, trajectories will be down-sampled to have at most \`max_points\` points. To avoid this, specify \`max_points=None\`.'

```
In [11]: tdf = skmob.TrajDataFrame(df, latitude='lat', longitude='lon', datetime='t', user_id='sourcemmsi'
         tdf.plot_trajectory(zoom=9, weight=3, opacity=0.9, tiles='OpenStreetMap', max_users=100)
```

Out[11]:

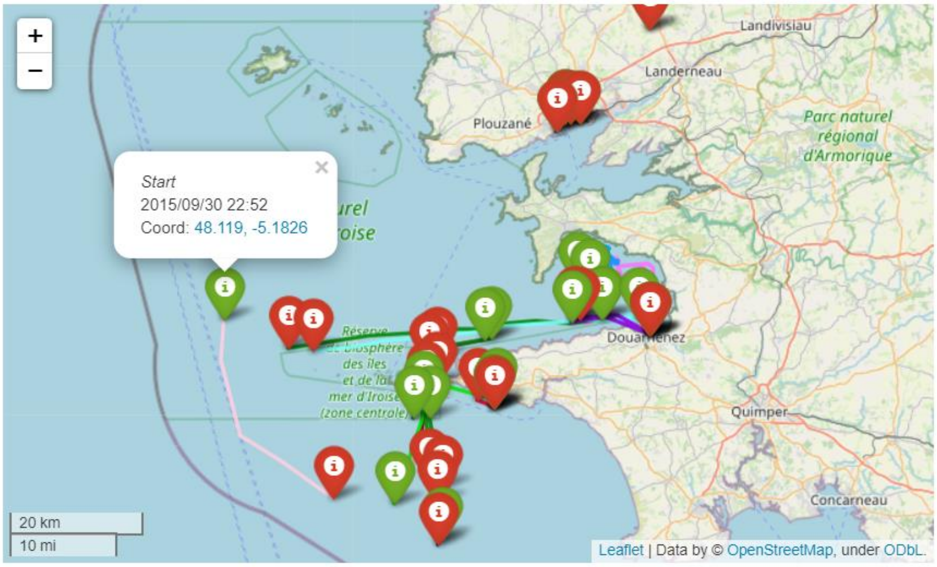


Figure 3: scikit-mobility / Folium plot of a TrajDataFrame with multiple trajectories and their start (green) and end (red) location markers superimposed on OpenStreetMap tiles.

Trackintel provides static plots using Matplotlib. The default plot for trip legs created from position fixes draws each trip leg in a colour indicating the moving object ID (user). As Figure 4 shows, in contrast to the previous libraries the default plot function does not provide background maps. Instead, the function provides a plot_osm keyword that 'will download an OSM street network and plot below the triplegs' (Trackintel documentation, 2022). This is certainly useful in the context of human movement in local urban environments, but it is not suitable for the ship movement dataset used in this example.

Traja provides different default plots for individual trajectories (TrajaDataFrame in Figure 5) and sets of trajectories (TrajaCollection in Figure 6). Plots for individual trajectories use coloured point markers to communicate the timestamp at a recorded location. In addition, grey lines are used to visually connect consecutive lines. Since our DataFrame contains data from multiple moving objects, the visualization in Figure 5 wrongly connects locations that belong to different moving objects. This issue is fixed by using a TrajaCollection, as shown in Figure 6. However, this changes the visualization, since the line and marker colours are now used to indicate the object ID. Therefore, all visual indicators of movement direction and speed are lost.

```
In [13]: pfs = ti.io.from_geopandas.read_positionfixes_gpd(gdf, tracked_at='t', user_id='sourcemmsi')
         pfs, sp = pfs.as_positionfixes.generate_staypoints(method='sliding')  # dist_threshold=10)
         pfs, tpls = pfs.as_positionfixes.generate_triplegs(sp, method='between_staypoints')
         tpls.as_triplegs.plot()
```

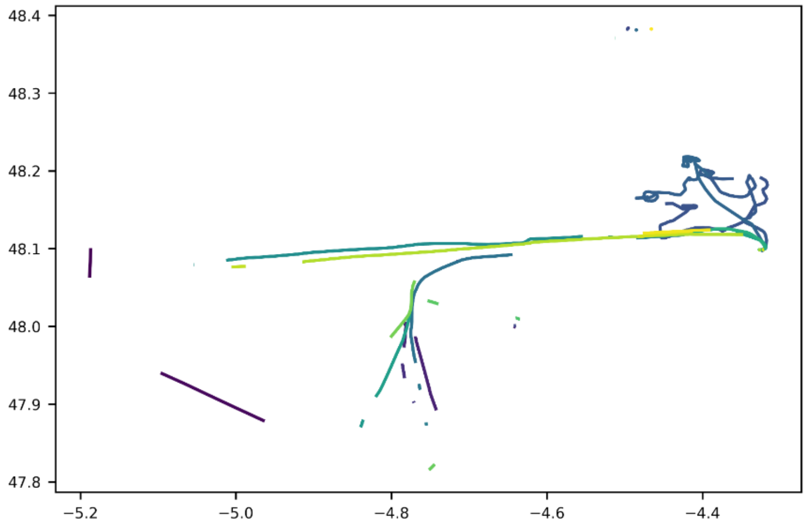


Figure 4: Trackintel / Matplotlib plot of trip legs computed from input position fixes, with colours indicating object ID.

```
trj = traja.TrajaDataFrame(df.rename(columns={"lon": "x", "lat": "y", "t": "time"}))
trj.traja.plot()
```

```
<matplotlib.collections.PathCollection at 0x1f47d3561f0>
```

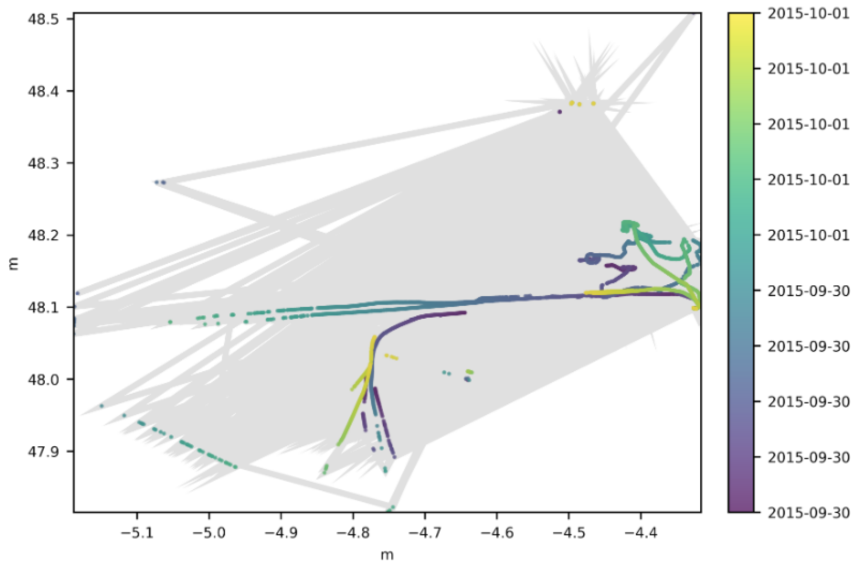


Figure 5: Traja / Matplotlib plot for individual trajectories, with colours indicating the time at locations along the trajectory.

```
[16]: trjs = traja.TrajaCollection(df.rename(columns={"lon": "x", "lat": "y", "t": "time"}), id_col='
      lines = trjs.plot()
```

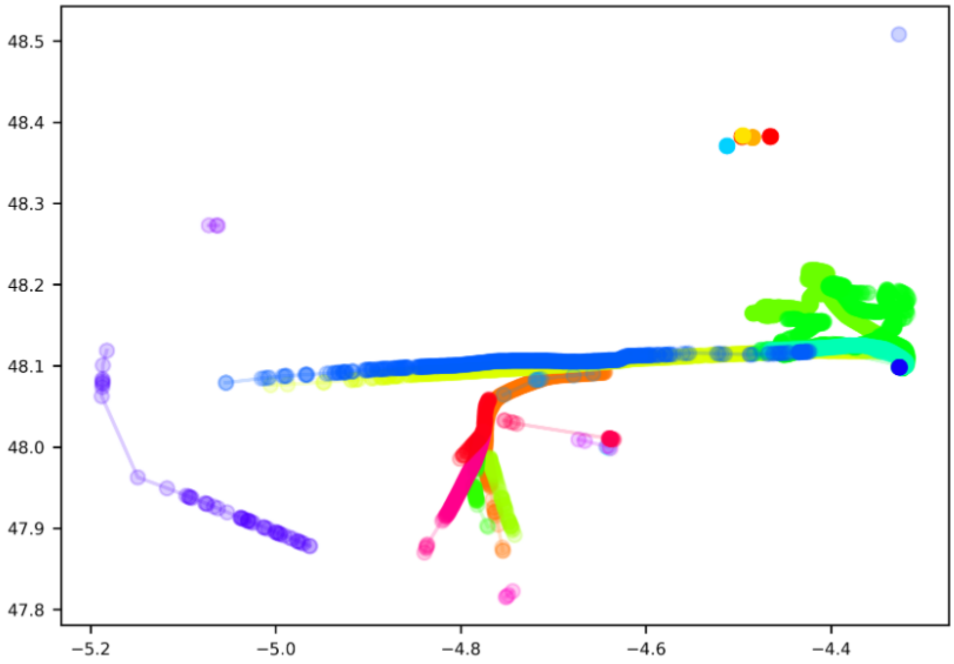


Figure 6: Traja / Matplotlib plot for sets of trajectories, with colour indicating object ID.

TransBigData provides static and interactive plots. Static plots (Figure 7) are based on GeoPandas and Matplotlib, since TransBigData's points_to_traj function returns a GeoDataFrame with LineString geometries. Users therefore need to specify the object ID column explicitly to obtain a plot with differently-coloured lines.

The interactive plots are based on Kepler.gl, which provides dedicated trajectory support, including animation capabilities. Trajectories can be coloured by a column. For the example in Figure 8, this column was manually set to the object ID. Another customization option is the trail length. This option defines how fast the trail fades out and thus provides a visual indication of the movement speed.

```
In [19]: traj_gdf = tbd.points_to_traj(df, col=['lon','lat','sourcemmsi'])
         traj_gdf.plot(column='sourcemmsi', categories=df.sourcemmsi.unique().tolist())

Out[19]: <AxesSubplot:>
```

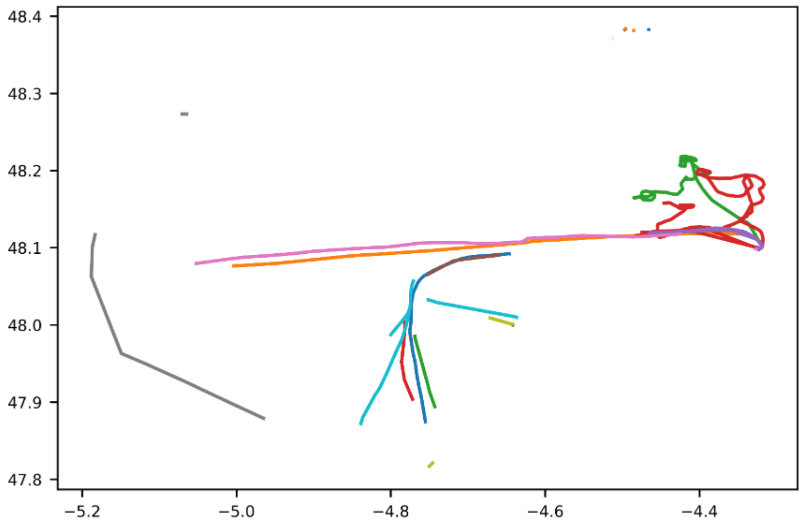


Figure 7: TransBigData / GeoPandas plot with colour indicating object ID.

```
[64]: tbd.visualization_trip(df, col=['lon','lat','sourcemmsi','t'])

      Processing trajectory data...
      Generate visualization...
      User Guide: https://docs.kepler.gl/docs/keplergl-jupyter
```

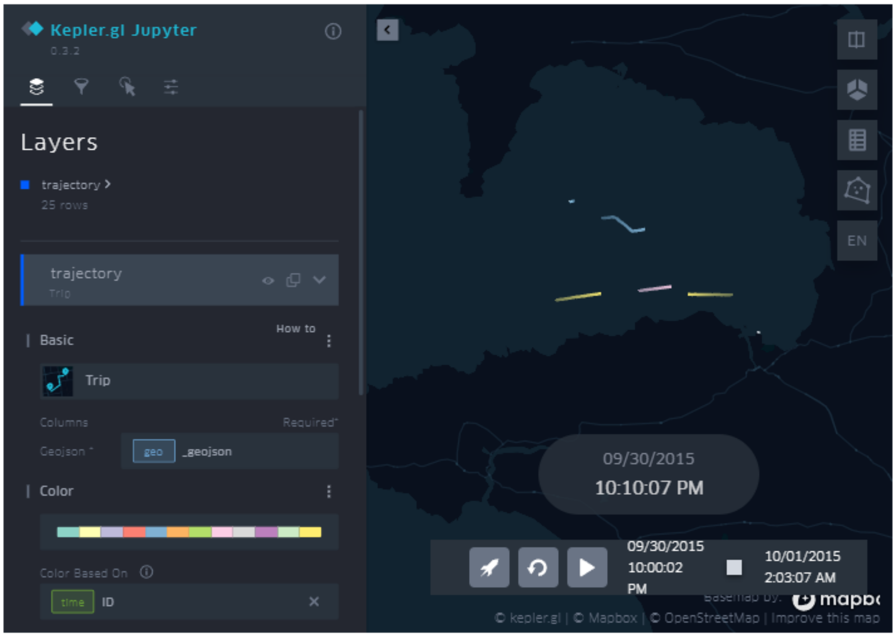


Figure 8: TransBigData / Kepler.gl plot provides animation capabilities.

The example plots shown in Figures 1–8 illustrate the choices made by developers of trajectory analysis libraries regarding the spatial visualization of trajectories.

Below is a list of the key decisions (PTRAIL is not included due to a lack of details in the library documentation, and library import errors in the official demo notebooks on Google Colab). Table 2 provides a summary.

1. *Interactivity.* It is noteworthy that not all development teams decided to implement interactive plots (Trackintel and Traja provide only static plots). Lack of interactivity limits the amount of information users can gain from the plots, since they cannot look more closely at details or bring up additional information in pop-ups.
2. *Background maps.* PyMove, MovingPandas and scikit-mobility provide geographic context through access to different map tile sources. This makes it possible to adapt the background maps to different analysis contexts. Trackintel can only plot the local OSM graph, which may be due to Trackintel's focus on human mobility data.
3. *Object identity* is commonly communicated using colour. Only TransBigData does not use colour to distinguish objects by default. Since colour is also used to communicate other properties, such as speed (MovingPandas) or time at location (Traja), users must choose one or the other.
4. *Movement direction* is communicated via markers (MovingPandas and Traja), colour (Traja) or animation (TransBigData).
5. *Speed* is not readily discernible from most plots. Only MovingPandas and TransBigData plots show speed, using colour and animation respectively.
6. *Time at location* is not readily discernible from most plots. Only Traja and TransBigData show time, using colours (only for plots of single trajectories) and animation respectively. Lack of time information makes it hard for users to distinguish whether two or more moving objects actually met (spatiotemporal co-location), or whether their trajectories only intersected spatially.
7. *Performance optimizations.* Static MatplotLib plots (Trackintel and Traja) have faster rendering performance than interactive plots (PyMove, MovingPandas and scikit-mobility). To avoid excessive rendering times, scikit-mobility developers have opted to automatically perform down-sampling (i.e. reduce the number of trajectories rendered and the number of points per trajectory), which can be customized by the user. PyMove and MovingPandas provide multiple trajectory generalization and down-sampling methods, but they are not applied automatically.

Table 1: Summary of the features of default visualizations created by trajectory analysis libraries

	PyMove*	MovingPandas	scikit-mobility	Trackintel	Traja	TransBigData
Interactivity	**Static & interactive**	**Static & interactive**	**Interactive**	Static	Static	**Static & interactive**
Background maps	**Misc tiles**	**Misc tiles**	**Misc tiles**	OSM graph only	no	**yes**
Object identity	**Colour**	**Colour1**	**Colour**	**Colour**	**Colour2**	Colour[3]
Movement direction	Start/end markers4	Start/end markers3	Start/end markers4	no	See time at location	Animation & tapered trail
Movement speed	no	**Colour**[1]	no	no	no	Animation & tapered trail
Time at location	no	no	no	no	**Colour2**	Animation
Performance enhancements	Misc generalization options3	Misc generalization options3	Downsampling4	no	no	no

* Based on the PyMove documentation (2022)
[1] Either object identity or speed
[2] Either object identity or time
[3] Manual
[4] Automatic

Of course, these choices do not necessarily reflect the full capabilities of the underlying spatial visualization libraries. Therefore, the following section highlights additional library functionalities that data scientists may find useful when dealing with movement data.

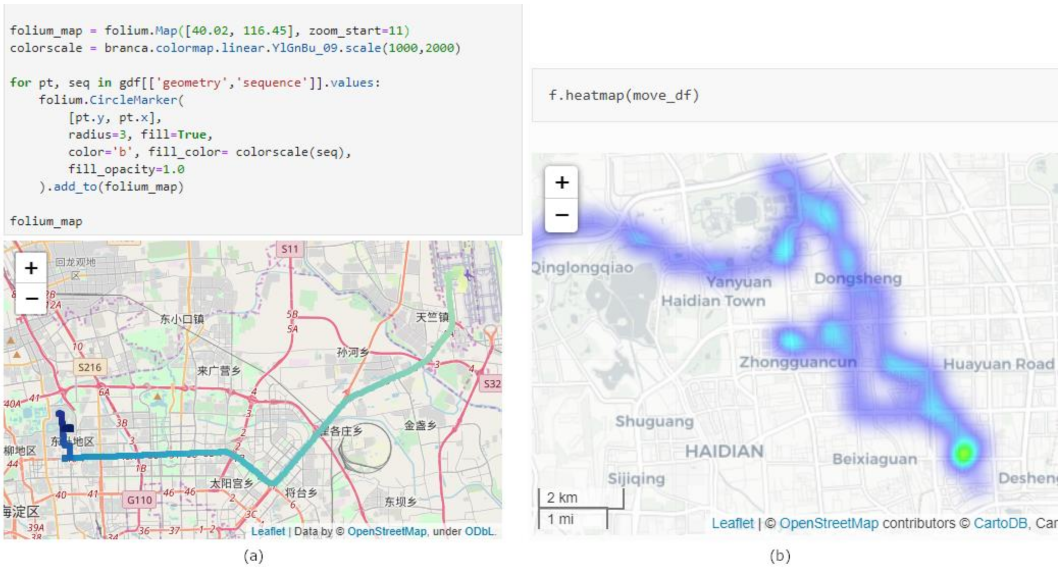
3.2 Spatial visualization libraries

Folium brings Leaflet-based maps into Jupyter notebooks and allows easy access to a variety of base maps to provide geographic context to movement data. Folium does not provide dedicated trajectory visualization support. However, marker or line colour and width are readily customizable once the trajectory is split into its individual elements, as shown in Figure 9a.

Many popular Leaflet plugins are accessible from within Folium[2], including the AntPath plugin for creating moving line patterns, Heatmap plugin (Figure 9b), PolyLineOffset plugin for drawing with an offset specified in pixels, without modifying the actual line coordinates, and the TimestampedGeoJson plugin. The Folium.TimestampedGeoJson plugin makes it possible to create animated trajectory visualizations, which have an interactive time slider, as shown in

[2] https://python-visualization.github.io/folium/plugins.html

Figure 10. While there are multiple Leaflet plugins for creating arrows, such as leaflet-arrowheads[3] or Leaflet.PolylineDecorator[4], these are not currently usable in Folium[5].



```
folium_map = folium.Map([40.02, 116.45], zoom_start=11)
colorscale = branca.colormap.linear.YlGnBu_09.scale(1000,2000)

for pt, seq in gdf[['geometry','sequence']].values:
    folium.CircleMarker(
        [pt.y, pt.x],
        radius=3, fill=True,
        color='b', fill_color= colorscale(seq),
        fill_opacity=1.0
    ).add_to(folium_map)

folium_map
```

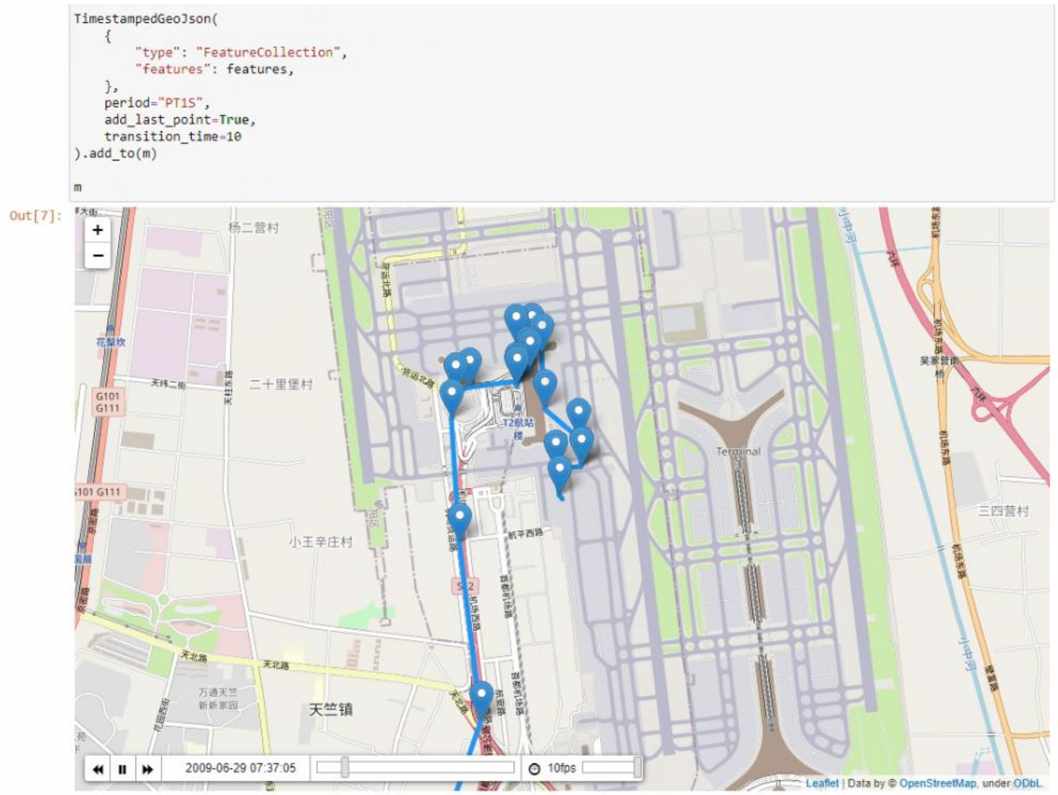
```
f.heatmap(move_df)
```

Figure 9: Folium in Jupyter (a) showing directionality using a colour gradient (source: own work), and (b) demonstrating heat map support, as implemented in PyMove (source: PyMove documentation, 2022).

[3] https://github.com/slutske22/leaflet-arrowheads

[4] https://github.com/bbecquet/Leaflet.PolylineDecorator

[5] https://github.com/python-visualization/folium/issues/1211



```
TimestampedGeoJson(
    {
        "type": "FeatureCollection",
        "features": features,
    },
    period="PT1S",
    add_last_point=True,
    transition_time=10
).add_to(m)

m
```

Out[7]:

Figure 10: Folium.TimestampedGeoJson plugin example trajectory visualization (source: own work). GeoViews is built on the HoloViews library and adds geographic plot types based on the Cartopy library, plotted using either Matplotlib or Bokeh. GeoViews supports different projections, as well as local coordinates without a defined coordinate reference system, as shown in Figure 11.

Like Folium, GeoViews does not provide dedicated trajectory visualization support, but line colour and width are readily customizable once the trajectory is split into its individual elements. In contrast to Folium, however, there is no functionality to apply a line offset[6].

Thanks to its tight integration with HoloViews, GeoViews plots can easily be combined with other plots to build flexible visualizations of multidimensional data. For example, Figure 12 shows a GeoViews-based MovingPandas trajectory plot and corresponding speed histogram that automatically updates when the trajectory generalization algorithm settings are changed using the linked tolerance value slider and drop-down list.

[6] https://github.com/holoviz/geoviews/issues/431

```
In [17]: get_file_from_url('https://github.com/anitagraser/movingpandas/raw/master/tutorials/data/soccer_field.png')

         pitch_img = hv.RGB.load_image('soccer_field.png', bounds=(0,0,pitch_length,pitch_width))
         (
             pitch_img *
             generalized.hvplot(title=title, c='team', colormap={'attack':'limegreen', 'defense':'purple'},
                                hover_cols=['player'],**hvplot_defaults) *
             generalized.get_start_locations().hvplot(label='start', color='orange')
         )
```

Out[17]:

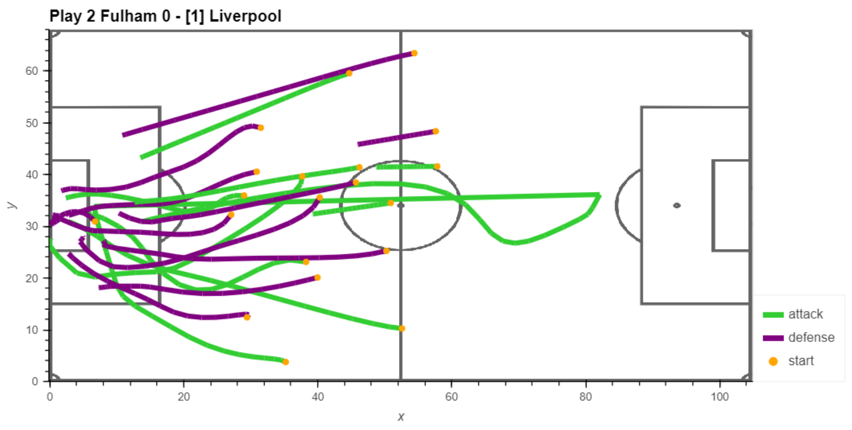
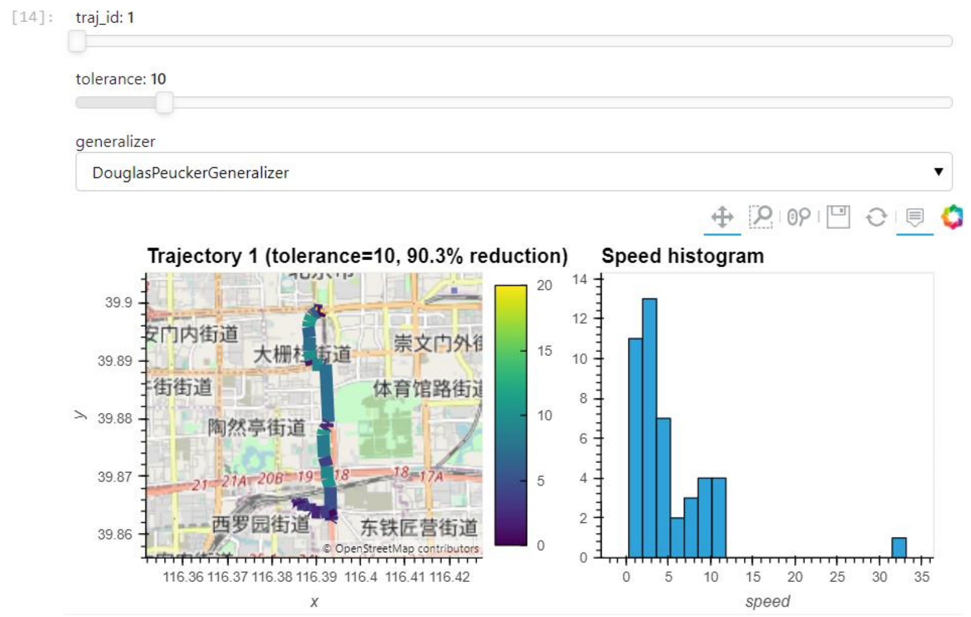


Figure 11: GeoViews-based trajectory visualizations, as implemented in MovingPandas using local coordinates referenced to a soccer pitch. (Source: own work)

```
[14]: kw = dict(traj_id=(1, len(traj_collection)), tolerance=(0, 100, 10), generalizer=generalizers)
      pn.interact(plot_generalized, **kw)
```



[14]:

traj_id: 1

tolerance: 10

generalizer

DouglasPeuckerGeneralizer

Figure 12: Interactive data exploration panel combining GeoViews-based trajectory visualization with a histogram of the corresponding speed values and options to change the trajectory generalization algorithm and tolerance setting. (Source: own work)

A related library in the HoloViews family is Datashader. Compared to Folium and GeoViews, Datashader can handle much larger datasets. This scalability is achieved by efficiently implemented rasterization routines that are distributed across CPU cores and processors using Dask, or GPUs using CUDA. This way, rasterized representations of the input dataset can be displayed in the notebook more quickly than, for example, vector-based Folium visualizations. It is particularly noteworthy that Datashader provides dedicated trajectory rendering support[7] that can be used to create trajectory visualizations, including point and line density maps, as shown in Figure 13 (Graser, 2021). However, there is currently no straightforward solution for colouring individual line segments based on attributes such as speed[8].

In [40]:
```
grouped = [df[['x','y']] for name, df in segment_df[segment_df.is_gap].groupby(['id', 'id_by_gap'
path = hv.Path(grouped, kdims=['x','y'])
plot = datashade(path, cmap=COLOR_HIGHLIGHT).opts(frame_height=FIGSIZE[1], frame_width=FIGSIZE[0]
BG_TILES * plot
```

Out[40]:

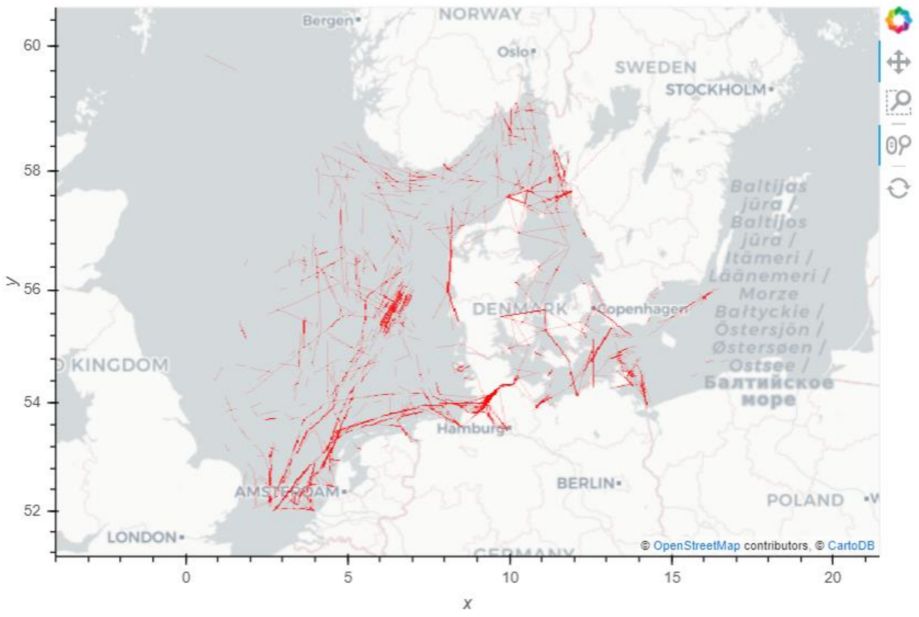


Figure 13: Datashader density map of trajectory line segments. (Source: own work)

To the best of our knowledge, the interactive visualization libraries presented so far do not provide 3D plotting features that would enable the straightforward implementation of space-time cubes, or similar 3D visualizations such as trajectory walls (Tominski et al., 2012).

To address the lack of space-time cube visualization tools in available movement analysis libraries, Tenkanen (2022) turns to the Plotly library, as shown in Figure 14, because of its

[7] https://datashader.org/user_guide/Trajectories.html

[8] https://github.com/holoviz/datashader/issues/969

easy-to-use API. It is worth noting, however, that this 3D plot lacks support for base maps, which may be considered an important component of space-time cubes. Other 3D solutions that may be used include, for example, D3.js and three.js, both JavaScript libraries that could be leveraged in ways similar to how Folium leverages Leaflet.

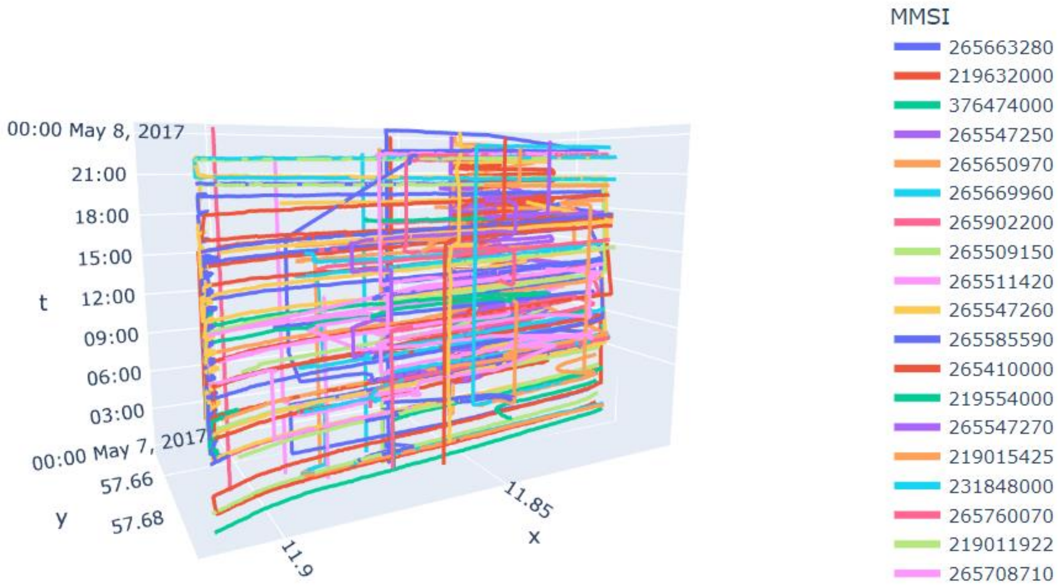


Figure 1: Space-time cube created with Plotly. (Source: Tenkanen, 2022)

4 Conclusions

In this paper, we have reviewed the current state of trajectory visualization tools in Jupyter notebook environments, covering seven movement analysis libraries that use four different spatial visualization libraries. Our review shows what the development teams of these movement analysis libraries chose to include in their spatial trajectory visualizations. These choices range from simple static plots to interactive and even animated plots that can visualize different trajectory properties, such as movement direction and speed, object ID, and time at a certain location.

Our review summary does not attempt to rank the movement analysis libraries based on their visualization capabilities. This was a conscious decision, because we perceive scientific software engineering not as a competition but as an opportunity to learn from each other. The development focus of many of the libraries reviewed clearly lies on their trajectory data processing capabilities rather than on trajectory visualizations. Since all the libraries reviewed are built on Pandas, there is considerable potential for integration, both on the developer side and on the user side, since DataFrames can be exchanged easily between libraries.

The movement analysis libraries we reviewed all depend on one or two of four visualization libraries: Matplotlib, Folium, GeoViews and Kepler.gl. Since Matplotlib and Folium are the default libraries of GeoPandas for static and interactive plotting respectively, it is not surprising that they are also commonly used to create spatial trajectory visualizations. However, we find that the cartographic capabilities of these libraries still exhibit some gaps that need to be filled to enable better trajectory visualizations for data science workflows, including data quality assessment. Particularly notable shortcomings include the limited line styling options (for example, missing capabilities to add arrowhead markers to visualize directionality, or capabilities to offset lines in order to reduce overlaps), as well as the general lack of visualization tools that make use of the third dimension (such as space-time cubes or trajectory walls). The only exception is Kepler.gl, which supports 3D plots and is used in one movement analysis library (but only to create animations). Other 3D plotting libraries, such as the JavaScript libraries D3 and three.js, have not been integrated so far.

Besides general cartographic capabilities, another important factor affecting the usability of visualization libraries is their performance or rendering speed. In trajectory visualizations, the number of individual features that have to be rendered grows quickly, particularly when each trajectory segment is drawn individually because we want to style the line according to the segment's speed or other property. Slow rendering performance causes undesirable wait times and therefore risks limiting acceptance by data analysts and scientists.

In addition to cartographic capabilities and rendering speed, developers of movement data visualization tools also have to strive for ease of use, since even the most powerful libraries will have trouble establishing a solid user base if data scientists find them too cumbersome to learn and use. To achieve a critical mass of users that can support sustainable development, future scientific software development work should ideally focus on bringing together the currently disjointed efforts that are spread over multiple different movement analysis and visualization libraries.

References

Andrienko, G., Andrienko, N., Chen, W., Maciejewski, R., & Zhao, Y. (2017). Visual analytics of mobility and transportation: State of the art and further research directions. *IEEE Transactions on Intelligent Transportation Systems*, *18*(8), 2232-2249.

Chen, W., Guo, F., & Wang, F. Y. (2015). A survey of traffic data visualization. *IEEE Transactions on Intelligent Transportation Systems*, *16*(6), 2970-2984.

Gahegan, M. (2020) Fourth paradigm GIScience? Prospects for automated discovery and explanation from data, *International Journal of Geographical Information Science*, 34:1, 1-21, DOI: 10.1080/13658816.2019.1652304

Graser, A. (2019). MovingPandas: Efficient Structures for Movement Data in Python. GI_Forum – Journal of Geographic Information Science 2019, 1-2019, 54-68.

Graser, A. (2021). An exploratory data analysis protocol for identifying problems in continuous movement data. Journal of Location Based Services, 15(2), 89-117.

Graser, A. & Dragaschnig, M. (2020). Open Geospatial Tools for Movement Data Exploration. KN – Journal of Cartography and Geographic Information, 70(1), 3-10. doi:10.1007/s42489-020-00039-y.

Haidri, S., Haranwala, Y. J., Bogorny, V., Renso, C., da Fonseca, V. P., & Soares, A. (2021). PTRAIL--A python package for parallel trajectory data preprocessing. arXiv preprint arXiv:2108.13202.

He, J., Chen, H., Chen, Y., Tang, X., & Zou, Y. (2019). Diverse visualization techniques and methods of moving-object-trajectory data: a review. *ISPRS International Journal of Geo-Information*, *8*(2), 63.

Oliveira, A. F. D. (2019). Uma arquitetura e implementação do módulo de visualização para biblioteca PyMove. Bachelor's thesis. Universidade Federal Do Ceará.

Pappalardo, L., Simini, F., Barlacchi, G., & Pellungrini, R. (2019). scikit-mobility: A Python library for the analysis, generation and risk assessment of mobility data. arXiv preprint arXiv:1907.07062.3

PyMove documentation. (2022). Retrieved from: https://pymove.readthedocs.io/en/latest/examples/03_Exploring_Visualization.html

Ray, C., Dreo, R., Camossi, E., & Jousselme, A.-L. (2018). Heterogeneous Integrated Dataset for Maritime Intelligence, Surveillance, and Reconnaissance (0.1) [Data set]. Zenodo. Retrieved from: https://doi.org/10.5281/zenodo.1167595

Rey, S. (2021), Geographical Analysis: Reflections of a Recovering Editor. *Geogr. Anal.,* 53: 38-46. https://doi.org/10.1111/gean.12193

Shenk, J., Byttner, W., Nambusubramaniyan, S., & Zoeller, A. (2021). Traja: A Python toolbox for animal trajectory analysis. Journal of Open Source Software, 6(63), 3202.

Tenkanen, H. (2022) Spatial data science for sustainable development course. Tutorial 3 - Trajectory data mining in Python. Retrieved from: https://sustainability-gis.readthedocs.io/en/latest/lessons/L3/mobility-analytics.html

Tominski, C., Andrienko, N., Andrienko, N., & Andrienko, N. (2012) Stacking-based visualization of trajectory attribute data. IEEE Trans. Vis. Comput. Graph. 2012, 18, 2565–2574.

Trackintel documentation. (2022). Retrieved from: https://trackintel.readthedocs.io/en/latest/modules/visualization.html

Yu, Q., & Yuan, J. (2022). TransBigData: A Python package for transportation spatio-temporal big data processing, analysis and visualization. Journal of Open Source Software, 7(71), 4021.