Trackintel: An open-source Python library for human mobility analysis

Henry Martin[1,2,*], Ye Hong[1,*], Nina Wiedemann[1,*], Dominik Bucher[3,**], Martin Raubal[1]

[1] Institute of Cartography and Geoinformation, ETH Zurich, Zurich, Switzerland
[2] Institute of Advanced Research in Artificial Intelligence (IARAI), Vienna
[3] c.technology, Tessinerplatz 7, 8002 Zurich, Switzerland
[*] Authors contributed equally. Order was determined randomly
[**] The majority of this work was done while the author was at the Chair of Geoinformation Engineering, ETH Zurich.

Abstract

Over the past decade, scientific studies have used the growing availability of large tracking datasets to enhance our understanding of human mobility behavior. However, so far data processing pipelines for the varying data collection methods are not standardized and consequently limit the reproducibility, comparability, and transferability of methods and results in quantitative human mobility analysis. This paper presents Trackintel, an open-source Python library for human mobility analysis. Trackintel is built on a standard data model for human mobility used in transport planning that is compatible with different types of tracking data. We introduce the main functionalities of the library that covers the full life-cycle of human mobility analysis, including processing steps according to the conceptual data model, read and write interfaces, as well as analysis functions (e.g., data quality assessment, travel mode prediction, and location labeling). We showcase the effectiveness of the Trackintel library through a case study with four different tracking datasets. Trackintel can serve as an essential tool to standardize mobility data analysis and increase the transparency and comparability of novel research on human mobility.

***K*eywords** Human mobility analysis · Open-source software · Transport planning · Data mining · Python · Tracking studies

1 Introduction

Human mobility studies using large-scale human digital traces have boomed over the last decade. On the collective level, researchers revealed that human movement can be universally described using statistical distributions, i.e., the power-law distribution of consecutive displacements [8, 44], stationary time between displacements [44, 52], and characteristic distance travelled by individuals (i.e., the radius of gyration) [20, 41]. Moreover, it has been shown that individuals exhibit markedly regular location visitation patterns [47] with high theoretical predictability [53]. People spend most of their time in a few locations [20, 52] and maintain a stable number of important locations over time [2].

To a large extent, this progress can be attributed to the widespread availability of large mobility datasets stemming from information and communications technology (ICT) and location based services (LBS) that are now integrated in many aspects of our daily life [26, 31]. Aside of the progress on the analysis of human movement itself, the increased availability of tracking data has led to a rapid growth of studies that use human mobility data to study phenomena related to human mobility, such as understanding of residential income segregation [39], quantifying urbanization levels and city livability [6], urban sensing [1], developing infrastructure for sustainable mobility [58] and responding to epidemic spreading [12]. However, the *raw* digital traces are often not the targeted unit of analysis; for example, a location where people perform an activity can not directly be derived from GPS track points or mobile phone tower data. Studies thus employ various steps to preprocess data into the desired format. These steps and their outcome are often different across studies [13] due to the variety of the datasets and the different understanding of the definitions, which has led to a vast collection of dataset-specific preprocessing and analysis methods. For example, the study by Feng et al. [17], which proposes the DeepMove model that is now widely accepted as a deep learning baseline model for next location prediction [35], generally regards each raw position record as a *location* and does not perform preprocessing.

However, focusing on the same problem, Urner et al. [55] extract *staypoints* (i.e., all the points where a user stayed for at least a certain duration) from GPS track points and further aggregate them into locations using the *k*-means algorithm. Solomon et al. [51] apply a similar processing concept but introduce the mean shift algorithm to detect staypoints, which are then merged into locations according to a distance threshold. These examples show how the lack of a standard movement model definition and a common preprocessing standard limit the reproducibility and comparability of the methods and analysis results.

To address these problems, we present Trackintel, an open-source python library for the processing and analysis of movement data. Trackintel is based on an established model for human mobility taken from transport planning, and implements the most important preprocessing steps. It further provides various analysis, visualization and support functions. Due to the versatility of the data model, Trackintel standardizes preprocessing steps for many types of tracking data that characterize the movement of individuals. It thereby greatly simplifies quantitative research based on tracking data and increases its reproducibility. The remainder of the paper is structured as follows. Section 2 provides an overview of existing libraries for the analysis and the preprocessing of movement data. Section 3 first introduces the hierarchical model for human mobility analysis and describes its implementation in Trackintel. This section then proceeds to present the most important functionalities of Trackintel to process movement data. In section 4 we showcase the capabilities of Trackintel to simplify the analysis and comparison of several different tracking datasets. Finally, we summarize and conclude this work in section 5.

2 Related work: Libraries for movement data

Due to the long history of research in transportation, human migration, and animal behavioral research, a large variety of libraries for (human) movement data processing exists. Joo et al. [29] survey an impressive number of 58 packages for movement analysis in R. Based on this work and the overview provided by Graser [22], we selected the libraries that aim at supporting movement analysis in Python, R and C++. In Table 1, these selected libraries are compared in terms of their user-friendliness (documentation and robustness), their focus and their provided functionality for human movement data analysis. To compare packages by the quality of their documentation, we evaluate them on a scale from 0-6 based on criteria used for peer-review of packages by pyOpenSci[1] and ROpenSci[2]. See appendix A for the list of criteria.

Many of the surveyed R libraries have a strong focus on animal behavioral analysis [29] (not all included in Table 1). The packages that can (also) be applied to human mobility analysis have a focus on basic statistical analysis of trajectories, such as measuring the spatial extent of animal motion (e.g., adehabitatLT [10]), or the duration and distance of movement trajectories (e.g., TrackR [18]). Currently there is no coherent framework available in R that provides the functionalities required for human movement analysis with a focus on transport applications. Furthermore, there are several libraries available in C++ such as Tracktable [57], MEOS[3], and MoveTK[4] that promise efficient and fast tools for trajectory data processing, although they may be less accessible for the research audience in human mobility and transportation. Furthermore, these libraries provide only highly specific functionalities and do not represent a comprehensive framework for movement data analysis.

ArcGIS Pro is a proprietary software for general spatial data processing with modules for movement data analysis such as speed and acceleration computation, trackpoint clustering and in particular trajectory visualization[5]. However, the different functionalities are scattered over different toolboxes and ArcGIS Pro does not provide a consistent framework for the analysis of movement data. Due to its proprietary nature, we could not evaluate documentation and testing as we did for the other packages, but we assume both are on a high level. We did not include QGIS[6], a high quality open-source GIS Project, in the table, as there are no well-maintained plug-ins for movement or trajectory data analysis available. However, QGIS could be used in combination with Python libraries or the mobilityDB [64] library.

In Python, many open-source libraries have emerged as tools to both facilitate and standardize data processing and analysis. The geographic information science (GIScience) community in particular has benefited significantly from Python libraries, for example, the data models implemented in Shapely [19] and the I/O formats for geographic data as

[1]https://www.pyopensci.org/contributing-guide/intro.html
[2]https://ropensci.org/
[3]https://github.com/adonmo/meos
[4]https://github.com/movetk/movetk
[5]https://pro.arcgis.com/en/pro-app/2.8/tool-reference/intelligence/an-overview-of-the-movement-analysis-toolset.htm
[6]https://www.qgis.org/en/site/

Table 1: Comparison of movement data libraries. Packages are predominantly available open source in R and Python and they are compared with regards to their focus, documentation and functionality. While other movement analysis libraries already provide well-maintained and documented code with rich functionality for trajectory analysis, only Trackintel provides robust and flexible methods to aggregate trajectories into locations, trips and tours.
(✓/ ✓̸/ x : available / partially available / not available)

Package name	**Focus**	Programming language Python (P)	Documentation score	Test coverage (* / **: not reported but low / high)	individual (I) / collective (C)	human (H), animal (A) and/or object(O)	Staypoint detection	Aggregation to location	Aggregation to trips	Aggregation to tours	Tracking quality assessment	Transport mode labelling	Home and work labelling	Visualization	Trajectory statistics (- / + / ++ : none / basic / rich)
Trackintel	Human mobility analysis	P	6	98%	I	H	✓	✓	✓	✓	✓	✓	✓	✓	+
Scikit-mobility [42]	Human mobility analysis	P	5	**	I	H	✓	✓	x	x	✓̸	x	✓̸	✓	++
Movingpandas [21]	Movement data analysis	P	6	96%	I	H/A/O	✓	x	✓̸	x	x	x	x	✓	++
PyMove [46]	Querying and visualizing trajectories	P	5	85%	I	H/A/O	✓	x	x	x	x	x	x	✓	+
MovinPy	Mobility data analysis	P	3	0	I	H	x	x	x	x	✓̸	x	x	x	-
HuMobi [50]	Human mobility prediction	P	3	0	I	H	✓	✓	x	x	✓̸	x	x	x	+
PTRAIL [23]	Parallelization and feature extraction	P	4	**	I	H/A/O	x	x	x	x	x	x	x	✓	++
TransBigData [59]	Transportation	P	5	90%	C	H	✓	x	x	x	✓̸	x	✓	✓	-
mobilityDB [64]	Storing and querying	SQL	6	97%	I	H/A	x	x	x	x	x	x	x	✓̸	+
Traja [49]	Animal trajectories	P	6	76%	I	A	✓̸	x	x	x	x	x	x	✓	++
Tracktable [57]	Moving object tracking	P/C++	2	**	I	O	✓̸	x	x	x	x	x	x	✓	++
MEOS	Spatio-temporal data analysis	C++	4	*	I	H/A/O	x	x	x	x	x	x	x	x	+
MoveTK	Computational movement analysis	C++	-	**	I/C	H/A/O	x	x	x	x	x	x	x	x	++
adehabitatLT [11]	Animal habitat	R	4	**	I	A	x	x	x	x	x	x	x	x	++
moveVis	Visualization	R	6	93%	I	A	x	x	x	x	x	x	x	✓	-
stplanr [34]	Sustainable transport planning	R	6	*	C	H	x	x	x	x	x	x	x	x	-
trajectories	Object tracking and interaction	R	5	*	I	O/H/A	x	x	x	x	x	x	x	✓	+
TrackR	Running and cycling data	R	6	52%	I	H	✓	x	x	x	x	x	x	✓	+
ArcGIS Pro	Spatial data	(P)	-	**	I/C	O/H/A	✓̸	✓̸	x	x	x	x	✓̸	✓	++

offered in the Fiona package[7]. Most importantly, spatial data can be handled easily with the Geopandas library [30] that directly builds up on Pandas [40], one of the most established Python libraries for data analysis and manipulation.

In the past years, Python has become the de-facto standard for data science and machine learning applications, which are increasingly important for the analysis of movement data [35, 54]. However, only few libraries have attempted to provide preprocessing and analysis tools specifically for human mobility in a comprehensive Python package (see Table 1). Although many algorithms for trajectory data mining were developed in the last decade [61], their open-source availability in Python is limited and they often suffer from insufficient documentation and testing standard, such as HuMobi [50] or MovinPy. Others are well-maintained but limited in scope, such as Traja [49] that targets animal movement, PTRAIL [23] for parallel processing, and TransBigData [59] which focuses on data analysis on a collective level, similar to the R library stplanr [34].

Notable exceptions are MovingPandas [21], scikit-mobility [42]. MovingPandas is based on Pandas and Geopandas, and focuses on low-level trajectory manipulation such as splitting, merging and visualizing trajectories. On the contrary,

[7] https://github.com/Toblerity/Fiona

the scikit-mobility library [42] targets high-level analysis functions, including computing human mobility metrics, generating synthetic trajectories and assessing privacy risks. Both libraries are actively maintained and contain various measures to ensure high code quality, but the definition of their data model implies a focus on movement trajectories (MovingPandas) or mobility flows (scikit-mobility), which omits important concepts describing individual human mobility such as activities, trips or tours [4].

We aim to close this gap with the Trackintel framework that utilizes an established data model from the transportation literature, which incorporates different semantic aggregation levels of tracking data specific to human mobility.

3 Trackintel framework

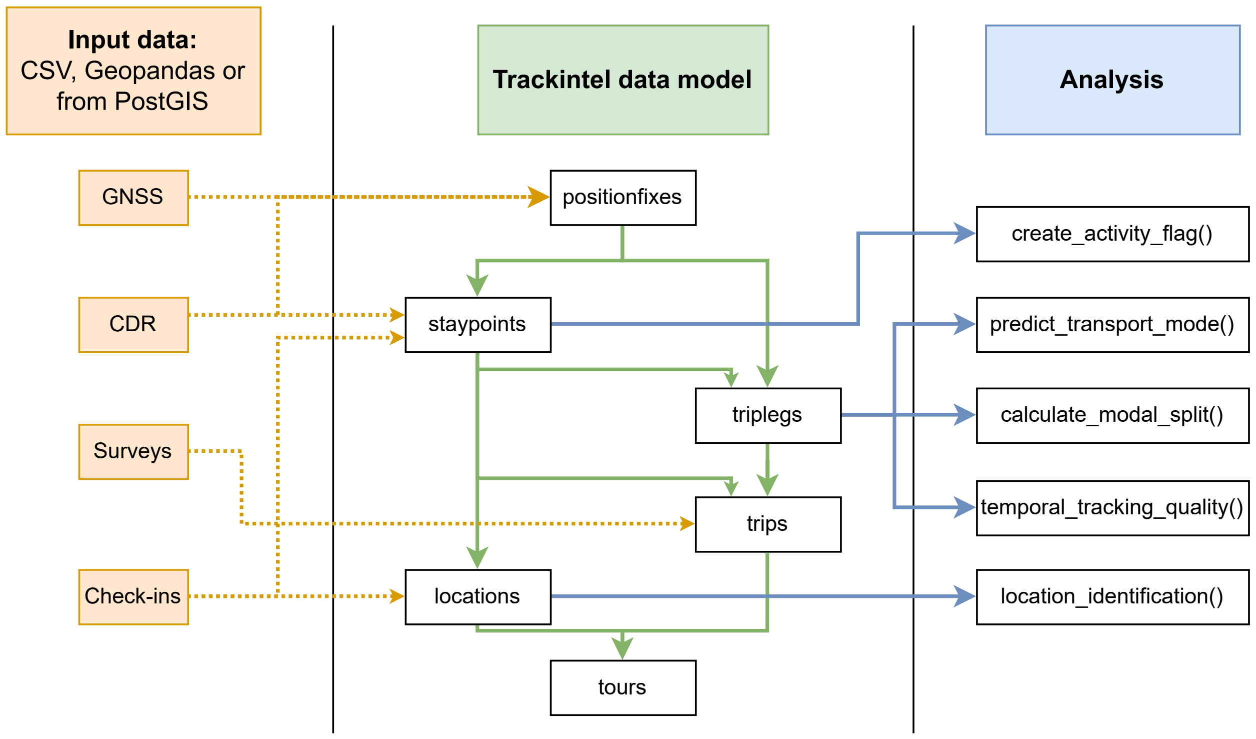


Figure 1: Overview of the Trackintel framework.

Trackintel[8] is a library for the analysis of spatio-temporal tracking data with a focus on human mobility. The core of Trackintel is the hierarchical data model for movement data [4] that is widely adopted in GIScience [9], transport planning [13] and related fields [45]. We provide easy-to-use and efficient functionalities for the full life-cycle of human mobility data analysis, including import and export of tracking data of various types (e.g, GPS trackpoints, location-based social network (LBSN) check-ins, call detail records), data model generation and preprocessing, analysis, and visualization. A conceptual overview of the different components of Trackintel can be found in Figure 1.

Trackintel focuses on the mobility of individual persons or objects (e.g., as opposed to crowd flows) and all functionalities are implemented as user-specific, based on unique user identifiers that link data to the respective tracked users. Trackintel is implemented in Python and is built mainly on top of Pandas [38] and GeoPandas [30] using accessor classes, a method to extend Pandas classes[9]. This design makes Trackintel easy to use for Python users and ensures its broad compatibility with other Python spatial analysis libraries.

3.1 The Trackintel data model

The modeling framework employed by Trackintel is based on the activity-based analysis framework in transport planning, which regards travel demand as derived from our need to perform activities at different locations. We

[8] https://github.com/mie-lab/trackintel
[9] https://pandas.pydata.org/docs/development/extending.html

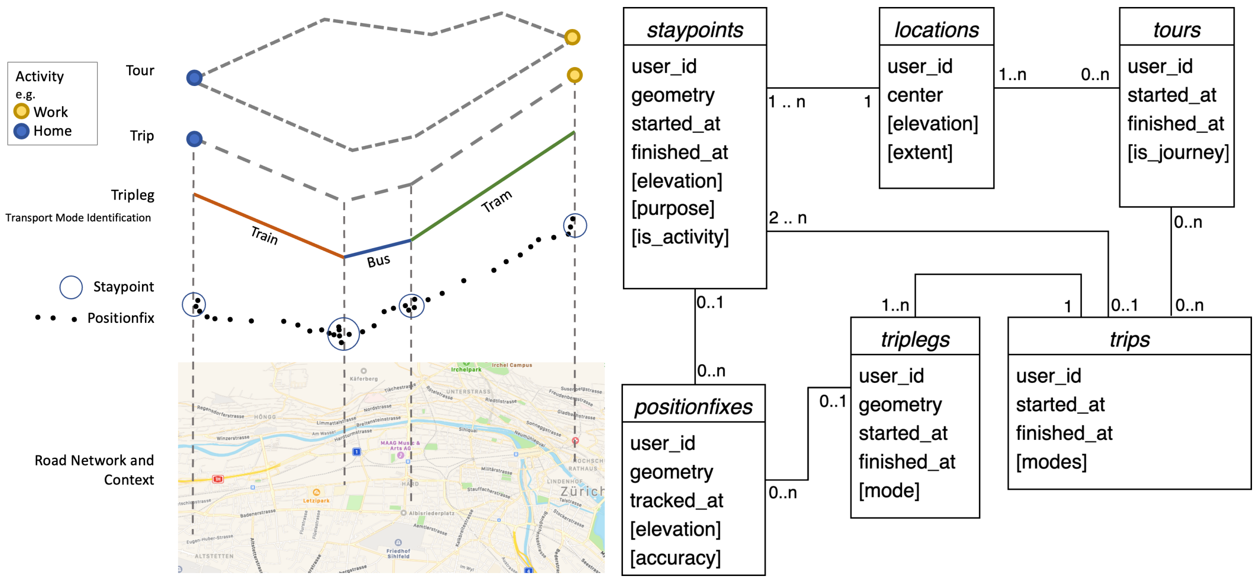


Figure 2: Semantic visualization of the Trackintel data models and their UML diagram, with mandatory and optional attributes (shown in square brackets). The relations between the different classes are shown in the connecting lines. Figure adopted from [28]

Table 2: Description of the mandatory and optional columns for Trackintel data models.

Data models	Fields	Description
All	id	The unique identifier for the record
	user_id	The unique user identifier
	tracked_at	The timestamp for the point (only for positionfix)
	started_at	The starting time of the record (except for positionfix and location)
	finished_at	The ending time of the record (except for positionfix and location)
Positionfix	geometries	Point geometry
Staypoint	geometries	Point geometry
	purpose (optional)	Purpose label for the staypoint. This could be either an activity purpose (e.g., home), or an non-activity purpose (e.g., wait).
	is_activity (optional)	Boolean flag indicating whether the staypoint is an activity
Location	center	Point geometry representing the center
	extent (optional)	Polygon geometry representing the extent
Tripleg	geometries	Line geometry
	mode (optional)	Transport mode label
Trip	origin_staypoint_id	The identifier of the starting staypoint
	destination_staypoint_id	The identifier of the destination staypoint
	primary_mode (optional)	The main transport mode label
Tour	location_id	The start and end location identifier
	journey	Boolean flag indicating whether the tour is a journey (A tour is called a journey if the start and end location is home).

follow the definition from [48] that people's daily mobility consists of staying at locations to perform activities and traveling between locations for the next activity [48]. In this definition and following the description in [4], movement is separated from activities at different semantic levels. Trackintel implements six classes to represent movement data in this hierarchical model: *positionfix*, *staypoint*, *tripleg*, *trip*, *tour*, and *location*. Figure 2 gives an overview of the hierarchical modeling structure and shows the classes in an UML diagram with their mandatory attributes and optional attributes in square brackets. All Trackintel classes are implemented as Pandas Dataframes or Geopandas Geodataframes. In order to be considered a valid Trackintel object, all mandatory attributes have to be present as columns with the correct names as shown in Figure 2. A more detailed explanation of the required and optional attributes of the Trackintel classes is given in Table 2. Geometries need to be of the defined type, with the exception of the *Location* class that can have multiple geometries. Furthermore, all timestamps for the time fields required by Trackintel have to be timezone-aware[10]. Besides these formal requirements, classes can contain any additional information required for specific analysis. In the following, the different classes and their semantics are introduced.

Positionfix. *Positionfix* is the smallest unit of tracking in the Trackintel data model, which consists of timestamped position records, for example generated by GNSS trackers or call detailed records (CDR) data. Positionfixes are often directly transferred from raw tracking data and are thus a natural entry point to the Trackintel data model, where it can further be processed and segmented into triplegs and staypoints. No inherent semantics are included since movements and activities cannot be distinguished from Positionfix.

Staypoint. *Staypoint* represents a point in space, which is defined as an individual remaining within a defined geographical radius for a defined time. Compared to the raw *positionfix* points, staypoints can represent stationary points that carry particular semantics, such as the purpose of the stay, or they can represent an intermediate stay such as waiting for a bus. To distinguish between these two types of staypoints, we introduce the concept of *activity*: an activity staypoint is usually the reason for a person to travel and has an important purpose with attached activity label (e.g., home), while a non-activity staypoint only represents a trivial stationary point (e.g., waiting). The exact definition of an activity depends on the goal of the study. In Trackintel, activities are staypoints with the attribute *activity_flag* set to *True*, which can be obtained through user labels or directly inferred from data (see section 3.5.2). While activity *staypoints* are the basic unit for constructing trips, which mark the start and end of a *trip*, non-activity *staypoints* can only be part of a *trip*. Additionally, *staypoints* can be spatially aggregated to form *locations*.

Tripleg. The most basic level of movement is defined as *tripleg* (referred to as stage in [4]), which formally represents a continuous movement without changing transport mode or vehicle. Therefore, triplegs contain semantics about the movement of an individual such as the mode of transport that is stored in the attribute field *mode* if available. This information can be obtained from user labels [25, 63] or inferred using heuristics directly from the data, which is implemented as labeling functions in Trackintel (see Section 3.5.1). Triplegs can be created from positionfixes and can be aggregated to form trips.

Trip. *Trip* represents all travels between two activities and summarizes all triplegs and non-activity staypoints between two consecutive activity staypoints. Trips inherit the activity purpose from the activity label attribute of the destination staypoint. As they are often the primary quantity of interest in transport planning studies, trips, together with activities, are the core of the movement data model proposed in Axhausen [4].

Location. Activity staypoints represent individual visits at places that are significant to the visitor. Trackintel models these significant places using the *location* class to enable the characterization of the place that is visited. While the information attached to staypoints is bound to the individual visit (e.g., the specific activity or the time of day), the semantics of locations are related to the place independent of the visit (e.g., land use or the opening hours of a shop). Locations are modeled with two different geometries, a point geometry for the center of the location and a polygon geometry to describe the extent of a location.

Tour. The mobility of individuals is centered around a few significant locations that act as the basis of their travel behavior. Individuals conduct several activities and trips if convenient but return home (or to a similar significant location) to plan their next activity. This behavior can be analyzed using the *tours* class, which is defined as "a sequence of trips starting and ending at the same location" [4, p. 4], referring to the location class defined above. A special case of a tour is the concept of *journey* that starts and ends at the home location of an individual. In Trackintel, a tour can be flagged as a journey using the *journey* attribute. A tour contains multiple trips, but one trip can also be part of several tours in case they are nested, e.g. the trip from the work location to the supermarket and back is part of a larger journey that started at home.

3.2 Data model generation

The core functionality of Trackintel is to generate all classes defined in the movement data model from the raw tracking data. In practice, this refers to the generation of the entire hierarchical movement data model from positionfix data. However, it should be noted that it is not required and often not practical to enter the framework from positionfixes - the framework can be accessed at any semantic level depending on the available data (e.g., location based social network (LSBN) check-ins represent staypoints without the availability of positionfixes; see Figure 1 for examples of input levels for different tracking data types). The following section presents the implemented preprocessing steps necessary to aggregate data through the hierarchy levels. The output of all `generate` functions is a (Geo)DataFrame with the fields listed in Table 2.

[10]See `https://docs.python.org/3/library/datetime.html#aware-and-naive-objects` for an explanation

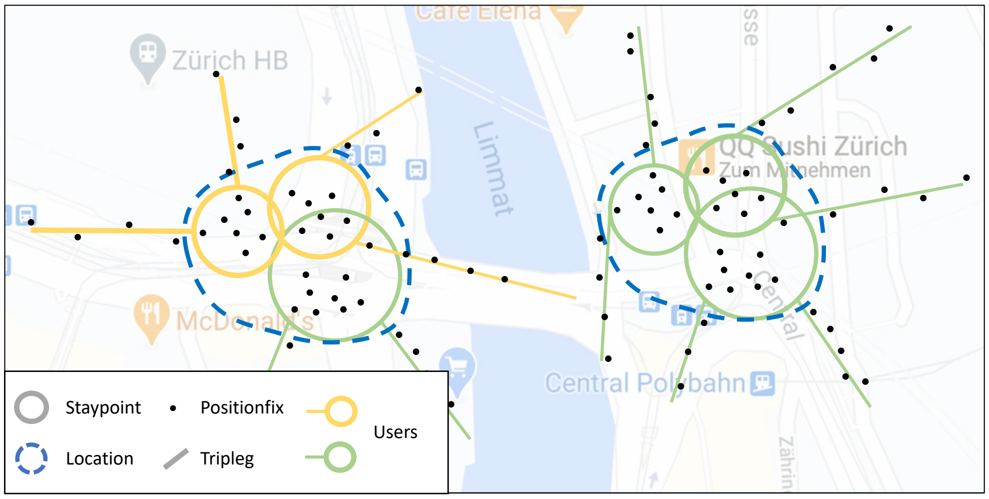


Figure 3: Semantic visualization of the relations between positionfix, staypoint and locations. Staypoints are groups of positionfixes where the users are stationary, and locations are aggregations of staypoints where the user visits multiple times. Locations can be generated across users (left) or for each user individually (right). Map data ©2022 Google.

Generate staypoints. In Trackintel, *staypoints* are generated from *positionfixes* using the `trackintel.preprocessing.positionfixes.generate_staypoints()` function. It implements an extended version of the sliding window algorithm for staypoint detection first reported in [33]. The function accepts predefined distance and time threshold parameters, iterates over all *positionfixes* and determines groups of points that satisfy the thresholds for each tracked individual. Therefore, each output staypoint inherits the starting and ending time from the first and last positionfix that belongs to it, respectively, as well as the mean geometry coordinates of the group of positionfixes. The implemented staypoint detection algorithm extends the algorithm from [33] by an option to exclude temporal gaps in the tracking data commonly observed in many datasets due to low temporal coverage. This behavior can be controlled using the *gap_threshold* parameter which represents the maximum time between two consecutive staypoints so that they are still considered consecutive.

Generate locations. Locations can be generated by aggregating staypoints spatially. We implement the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [28, 24] in the `trackintel.preprocessing.staypoints.generate_locations()` function to aggregate spatially-close staypoints into locations. DBSCAN adopts a set of neighborhood characterization parameters ϵ and $min\_samples$ to depict the tightness of the sample distribution and determine the clustering result. In the context of location generation, ϵ controls the distance of which nearby staypoints will be merged into a single location, and $min\_samples$ determines the minimum number of staypoints to form a location (i.e., how many visits are required at the same place to consider it as significant). Generated locations are equipped with two different geometries. The *center* of the location is a point geometry, calculated as the mean coordinates from all staypoints assigned to the cluster; the *extent* of the location is a polygon geometry, defined as the bounding box of all belonging staypoints. Furthermore, we provide the flexibility in the function to generate locations that are significant to a single user (Figure 3 right) or to all users present in a dataset (Figure 3 left). While user locations regard staypoints of each tracked user separately in the clustering process and prevent generating locations that are excessively large [3], dataset locations consider all staypoints at the same time and could output locations with shared semantics across users (e.g., train stations or shopping malls). In both options, the center and the extent of the clustered staypoints are attached to the generated locations, providing geometry information that facilitates further processing and analysis tasks.

Generate triplegs. Trackintel implements an algorithm that extracts triplegs from positionfixes based on the assumption that an individual is moving if he or she is not stationary, meaning that all positionfixes that do not belong to any staypoint are assigned to a tripleg. This assignment process is implemented in the `trackintel.preprocessing.positionfixes.generate_triplegs()` function, requiring as input the positionfixes with the identifier of the already generated staypoints. Internally, the function aggregates all positionfixes between two consecutive staypoints to form a tripleg, whose line geometry is constructed from connecting the chronically ordered point geometries. Similar to the generation process of staypoints, the start and end timestamp of each tripleg is inherited from the first and last positionfixes that belongs to it, respectively.

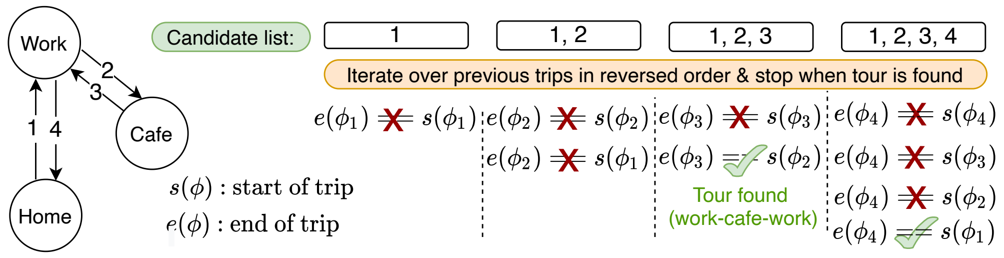


Figure 4: The algorithm of tour generation implemented in Trackintel. A list of start candidates is maintained and iteratively checked for tour-closing trips.

Generate trips. Trackintel implements the method `trackintel.preprocessing.triplegs.generate_trips()` to generate trips based on existing staypoints and triplegs. Trips summarize all movement and all non-essential actions (e.g. waiting) between two staypoints that are flagged as activity. The main result of the trip generation is the identifier management that connects trips with their associated staypoints and triplegs. Trips receive the fields *origin_staypoint_id* and *destination_staypoint_id* that point to the activities that started and ended the trip, respectively. Furthermore, the function adds the field *trip_id* to triplegs and non-activity staypoints that occur during a specific trip, and the fields *prev_trip_id* and *next_trip_id* to activity staypoints that are at the start or the end of a trip.

The trip detection that is implemented in Trackintel can handle incomplete tracking data and supports the detection of temporal gaps. A temporal gap is defined as missing tracking signals longer than a certain time period [60], which can be specified using the *gap_threshold* input parameter to the function. If a temporal gap greater than *gap_threshold* is detected, we assume the individual performed an unobserved activity and, therefore, the destination of the current and the origin of the next trip is unknown (NaN in the resulting table). Finally, the function provides the flexibility to specify whether the trips table should include the geometry using the argument `add_geometry`. The geometry of a trip consists of the points for the origin and destination staypoints. If the origin is unknown, we use the first point of the first tripleg instead, or analogously the last point for the destination.

Generate tours. To the best of our knowledge, there is no standardized approach yet how to combine trips to tours. Here, we take a rather broad definition of tours that includes nested tours as described in [4], leaving the user the choice to filter the outputs later. An example of a nested tour is shown in Figure 4: the tour Work-Cafe-Work is part of the longer tour Home-Work-Cafe-Work-Home. This definition implies an n-to-n relationship between trips and tours: One tour contains multiple trips, and one trip can be part of multiple tours. This is reflected in the output of the tour generation algorithm (`trackintel.preprocessing.trips.generate_tours()`), where the output table *tours* contains a list of trips as a field, and the table *trips* has a list of tour IDs for each trip.

Our algorithm to generate tours from trips is explained visually in Figure 4. We iterate over the trips that are sorted chronologically, and maintain a list of tour-starting candidates. Each trip ϕ_i is a potential candidate to start a tour. At each iteration, that is, for each trip, we first check whether there is a spatial gap between this and the previous trip ϕ_{i-1}. Two options are implemented: If the table *staypoints* with the attribute `location_id` is provided, we compare the location ID of the end of ϕ_{i-1} to the one of the start of ϕ_i, formally $l(e(\phi_{i-1})) = l(s(\phi_i))$. Alternatively, if the staypoints are not available, the predefined spatial distance threshold *max_dist* controls the maximum distance between the end- and start points, i.e. $d\big(e(\phi_{i-1}), s(\phi_i)\big) \leq max\_dist$.

Additionally, our implementation offers the possibility to generate tours that are partially observed to accommodate tracking datasets with a low temporal tracking coverage, e.g., mobile phone data-based studies. A parameter *max_nr_gaps* determines how many spatial gaps are allowed within a single tour. Note that no gaps are allowed at the start or end of a tour, because a tour must start and end at the same location, or the start- and end-staypoints must lie within the permitted range. If the test described above yields a spatial gap between ϕ_{i-1} and ϕ_i, and *max_nr_gaps*$= 0$, the candidate list is reset to $[\phi_i]$. Otherwise, a gap is registered.

Next, it is tested whether ϕ_i concludes a tour. For this purpose, we iterate over all candidates in the reversed order, such that the shortest possible tour is found first. Thereby we compare the start point of a candidate ϕ_j to the end point of ϕ_i. Again, the points are compared either by the location ID or via the *max_dist* parameter. If they are the same, the trips $\{\phi_k \mid j \leq k \leq i\}$ form a tour, subject to two further conditions: A. While iterating over candidates, the encountered gaps are counted and the time duration is checked. The parameter *max_time* is used to certify whether the tour takes place within an appropriate time period, by default 24 hours. B. When encountering more than *max_nr_gaps* in the reversed iteration, or when reaching a candidate that started more than *max_time* hours ago, the loop ends and no tour is found. Figure 4 shows an example where two tours are found after considering ϕ_3 and ϕ_4 respectively.

3.3 Import and Export

Reading and writing data are important steps in a standard movement data analysis pipeline. To simplify this process, Trackintel provides an I/O module for accessing movement data and storing intermediate or final results to a file or database. Three methods for converting movement data with attached attribute information to Trackintel-compatible formats are provided: Reading from Pandas Dataframes and Geopandas Geodataframes, from csv file formats and from PostgreSQL databases with PostGIS extension. Also, Trackintel implements helper functions to directly load tracking data from publicly available open-source datasets. The following describes these functions in details.

Geopandas. The recommended way to load new data into the Trackintel framework is via the Pandas/Geopandas reading functions. The Trackintel `trackintel.io.from_geopandas` module provides a read function for every Trackintel datatype that accepts GeoDataFrames as input. The functions support renaming and timezone conversion and return a valid Trackintel GeoDataFrame. Through these interfaces with Geopandas, Trackintel can process movement data stored in the most common geospatial file formats (e.g., shapefile, GeoJSON, and GeoPackage).

csv file formats. The Trackintel `trackintel.io.file` module can be used to read and write movement data with a csv file. The read functions provide a mapping of the column names to the required Trackintel attributes, and they transform and check the necessary geometry. The write functions provide an easy way to store Trackintel movement data to the disk for caching results or for distributing results.

PostGIS. Interfaces to access a PostGIS database are defined in Trackintel in the module `trackintel.io.postgis`, which allows to store or read datasets located in a PostgreSQL database with PostGIS extension using SQL. This enables the use of Trackintel for larger movement data sets.

Dataset readers. Trackintel additionally provides reading functions for transforming well-known public movement datasets from their raw data representation into the Trackintel data model. As an example, the Trackintel `trackintel.io.dataset_reader.read_geolife()` function reads the raw data from the Geolife dataset [63] and transforms them to positionfixes. The `trackintel.io.dataset_reader.geolife_add_modes_to_triplegs()` function adds the transport mode labels to triplegs, which are provided separately for some individuals in the Geolife dataset. The dataset reading functions facilitate and standardize the processing of public movement datasets using Trackintel, which also helps to benchmark new methods on the same data as related work.

3.4 Pre- and postprocessing

Trackintel offers several pre- and postprocessing methods such as the simplification of triplegs using the Douglas-Peucker algorithm [15] (the function `trackintel.preprocessing.triplegs.smoothen_triplegs()`) or the aggregation of consecutive staypoints (the function `trackintel.preprocessing.staypoints.merge_staypoints()`). The later is a common artifact in tracking data of various sources, in which several consecutive staypoints are generated during a single visit to the same location (e.g., due to noise or outliers recordings in GNSS tracking data). The function can aggregate staypoints that are visited close in time. Specifically, we propose to merge two staypoints s_1, s_2 of one individual if the following conditions hold: a) s_1 and s_2 are consecutive in time, b) s_1 and s_2 are assigned to the same location, c) there is no tripleg registered between s_1 and s_2, and d) the time gap between the end time point of s_1 and the start of s_2 is shorter than a predefined threshold $\theta_{max\_time\_gap}$. The input arguments to this method are thus the staypoints (with location id) and triplegs, as well as the threshold $\theta_{max\_time\_gap}$. An optional dictionary can be passed to specify how to aggregate the attributes that cannot be simply accumulated, e.g. the determination of the geometry of the aggregated staypoint.

3.5 Analysis

While the main functionality of Trackintel is the implementation of the hierarchical data model, the framework also includes advanced analysis functions in order to label transport modes, activity purposes and to assess the tracking quality of each individual.

3.5.1 Mode labeling

Applications in transport planning often require access to the travel modes of an individual [32]. Trackintel offers the function `trackintel.analysis.labeling.predict_transport_mode()` to impute the transport mode labels for triplegs. Since Trackintel does not assume the availability of user-provided labels, context or advanced data from the tracking device (e.g., accelerometer), we implement a simple heuristic to determine the travel mode from the tracking data. This classification is done per *tripleg* based on speed. The speed is approximated by the tripleg length (the distance of individual points in its LineString geometry) divided by its total time duration. The triplegs are labeled based on a simple division into slow mobility (<15km/h average speed), motorized mobility (<100km/h) and fast mobility (>100km/h). In future versions, a more in-depth analysis of travel patterns or map matching [27, 56, 5, 43] could be incorporated in Trackintel.

3.5.2 Location labeling

An individual's home- and work-locations play a major role for mobility data analysis. As described in Section 3, staypoints may be associated with an activity label, but oftentimes this information is not available. We assign "home" and "work" activity labels to the staypoints with an adapted version of the OSNA algorithm proposed by Efstathiades et al. [16]. The function, provided in `trackintel.analysis.location_identification.location_identifier()`, divides weekdays into rest, work and leisure time frames. The location with the longest accumulated duration in the "rest" and "leisure" periods is labeled as home, while work is set to the most predominant location in the "work" periods. While the original algorithm derives the hours spent at a location from geo-tagged tweets, we take advantage of the *started_at* and *finished_at* attributes of a staypoint. Additionally, similar as in the *R* package proposed by [14], we provide a fast method that simply assigns home and work labels to the two locations that are visited more often in the data (in this order). In both cases the locations can optionally be pre-filtered in order to exclude locations with an insufficient number of staypoints or an insufficient length of stay.

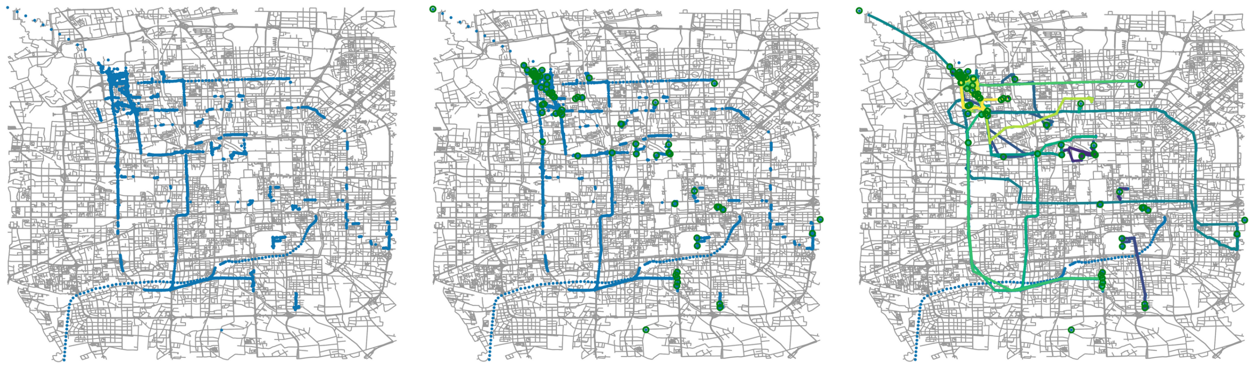
3.5.3 Modal split

If mode labels for *triplegs* are available, Trackintel supports the calculation of the modal split of travel. The Trackintel function `trackintel.analysis.modal_split.calculate_modal_split()` offers three options: Computing the modal split by count (i.e., how many triplegs with this mode exist), by duration (i.e., sum of individual's tripleg duration) or by travelled distance. Furthermore, the frequency can be set according to the Pandas time series frequency syntax[11]. The Boolean argument *by_user* determines the aggregation level, i.e. providing the modal split by user or by dataset. An example for one user is visualized in Figure 6 where the differences between a modal split by count (Figure 6a) and by distance (Figure 6b) stand out.

3.5.4 Tracking quality assessment

The function `trackintel.analysis.tracking_quality.temporal_tracking_quality()` is implemented to assess the temporal tracking quality of a given movement dataset. Temporal tracking quality, here defined as the proportion of time where the user's whereabouts are recorded, is regarded as a basic measure of the temporal resolution of the dataset [2]. The implemented Trackintel function is able to calculate the daily, weekly or overall tracking quality of each user according to the required granularity levels, which enables individual-level temporal resolution assessment, providing support for filtering low-quality users for further analysis. Additionally, tracking quality of hours of the day and weekdays can be obtained for measuring the tracking data quality differences across time periods.

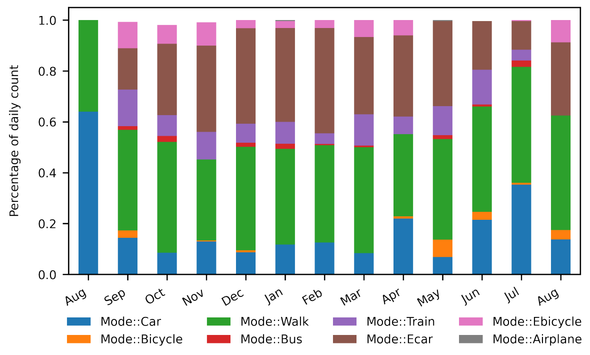
3.6 Visualization



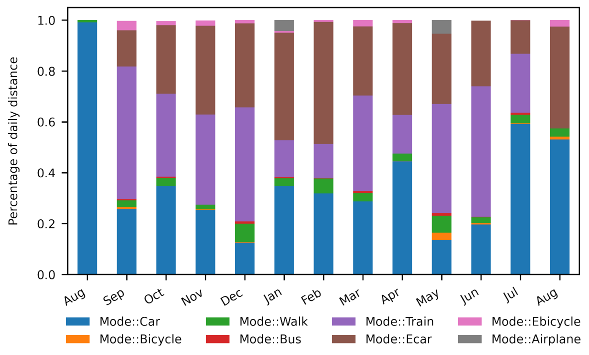
(a) Positionfixes (b) Staypoints (c) Triplegs

Figure 5: The Trackintel framework offers functions to plot positionfixes (a), staypoints (b), triplegs (c) together with the road network acquired from OpenStreetMaps. This example maps the movements of one Geolife participant.

[11]https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html



(a) Modal split by count



(b) Modal split by distance

Figure 6: The visualization result of the Trackintel `plot_modal_split()` function of the triplegs recorded from one Geolife participant. Major differences can be observed between the aggregation by count (number of triplegs) (a) and distance traveled (b).

Trackintel provides a `trackintel.visualization` module that supports the visualization of *positionfixes*, *staypoints* and *triplegs*. Our implementation standardizes these functions such that each data type can be displayed together with lower aggregation levels (see Figure 1). For example, the function `plot_locations()` has the optional arguments *positionfixes* and *staypoints*, such that all three can be shown together. In that case, *staypoints* and *locations* are displayed as circles with a predefined radius. Furthermore, Trackintel integrates osmnx [7] to optionally show the street network from Open Street Maps as background. Figure 5 shows example outputs of the `plot_positionfixes()`, `plot_staypoints()` and `plot_triplegs()` functions for one exemplary participant in the Geolife study.

Finally, Trackintel also provides a flexible method to visualize the development of the modal split over time. The `plot_modal_split()` function shows the output of the `calculate_modal_split()` function from the *analysis* module in a bar plot. Different temporal resolutions (i.e., weeks and months) are handled internally. An example for one user is shown in Figure 6 where the modal split has been aggregated by month.

4 A case study on multiple tracking datasets

Trackintel is a framework to standardize mobility data processing and analysis. To demonstrate its capability to handle data from various tracking studies, we provide a case study on four datasets. We read all data from a PostGIS database with the I/O module, preprocess them according to the Trackintel movement data model and compare the datasets in terms of tracking quality, trip characteristics, modal split. The code of the case study is available in the supplemental material and in the public repository[12].

4.1 Tracking studies

We include the data from four tracking studies with two different tracking data types. An overview of the dataset properties is given in Table 3. The first study is the open-source Geolife dataset [62] that covers the movement of employees of Microsoft Research Asia, who recorded their movement using GPS trackers. Second, we include two studies that were conducted in collaboration with the Swiss Federal Railway Systems (SBB) under the project name *SBB Green Class* [36]. In both studies, participants were given full access to all public transport in Switzerland. In addition, the participants from the first Green Class study (Green Class 1) received an electric vehicle and the ones from the second study (Green Class 2) an e-bike. Study participants were tracked with a GNSS-based application (app) called *Myway*[13]. The app already provides the data partially preprocessed as staypoints and triplegs. The same app was further used in our fourth dataset, the yumuv study[14] which investigated the impact of a Mobility-as-a-Service app that integrates shared e-scooters, e-bikes and public transport [37]. In the yumuv study, participants were divided into control and treatment group and were tracked for three months.

[12]`https://github.com/mie-lab/trackintel/blob/master/examples/Trackintel_case_study.pdf`
[13]`https://www.sbb.ch/en/timetable/mobile-apps/myway.html`
[14]`https://yumuv.ch/en`

Table 3: Overview of basic features of the considered tracking studies. Locations, staypoints, triplegs, trips and tours are given in multiples of a thousand.

	Users	Tracking period in days (std)	Input	Study type	Locations	Staypoints	Triplegs	Trips	Tours
Green Class 1	139	401 (59)	Staypoints, Triplegs	GNSS (app)	104.5	326.9	465.2	241.8	95.0
Green Class 2	50	314 (76)	Staypoints, Triplegs	GNSS (app)	35.7	87.9	128.6	61.4	22.7
Yumuv	806	87 (38)	Staypoints, Triplegs	GNSS (app)	127.3	326.3	502.3	199.7	83.0
Geolife	177	193 (443)	Positionfixes	GPS tracker	13.6	28.9	30.2	30.2	7.2

4.2 Standardized processing according to the Trackintel data model

The Trackintel framework offers a straightforward way to transform all data to the same format and to aggregate the data into trips and tours with minimal code. First, the raw GPS data in the Geolife dataset are converted to staypoints and triplegs with the Trackintel `generate_staypoints()` and `generate_triplegs()` functions. Staypoints are created with a distance threshold of 100m, i.e. a user must have traveled 100 meters to generate a new staypoint, and a temporal threshold of 30 minutes, as suggested in the original paper [33]. Furthermore, consecutive positionfixes with a temporal gap of more than 24 hours in between cannot belong to the same staypoint.

All further preprocessing steps based on staypoints and triplegs are applied with *the same* parameters for all four datasets. This ensures comparability of the results across datasets. More specifically, we derive the user's locations from the staypoints with the `generate_locations()` function. The method uses the DBSCAN algorithm with $\epsilon = 30$ meters and $min\_samples = 1$, such that one staypoint is sufficient to form a location. Furthermore, triplegs and staypoints are aggregated to trips with the `generate_trips()` function, with input parameter $gap\_threshold = 25$ minutes. At last, tours are generated by merging trips based on a maximum distance (*max_dist*) of 100m between their start and end points, and with the default parameters *max_nr_gaps*=0 and *max_time*=24 hours.

Table 3 provides the absolute numbers of locations, staypoints, triplegs, trips and tours per dataset. These quantities decrease from triplegs to trips and tours due to the aggregation steps. Note that for Geolife our parameter choices prevent triplegs from being merged (see Table 3 where the number of triplegs and trips are the same); however, parameters that are more suitable for the trip generation would have decreased the quality of other parts significantly due to the low tracking quality of Geolife. In total, the considered datasets include 769,957 staypoints and 1,123,931 triplegs. These quantities depend on the number of participants in the study and the total tracking duration. While the yumuv study has the largest sample size of 806 users, the Green Class 1 study participants have the longest tracking period, with each individual tracked for more than a year on average.

4.3 Analysis and comparison of tracking datasets

We now compare the mobility behavior of the study participants of all studies on the trip level as an exemplary usage of the Trackintel *analysis* module. The insights from this analysis are summarized in Table 4. First, we can derive the number of daily trips per individual from the absolute numbers given above. The study participants in Green Class 1 and Green Class 2 are most active in conducting trips. The low number of trips for Geolife users may be due to low temporal tracking coverage of the dataset. Furthermore, we compare the average trip distances and duration across datasets. Interestingly, yumuv and Geolife users take longer trips on average in terms of duration. There is also a clear effect of the bias of yumuv participants towards urban areas, where the trips cover much shorter distances. The number of trips per tour and the number of triplegs that are part of the same trip do not differ much between studies.

Table 4: Overview of the mobility statistics for the considered tracking datasets.

	Trips per day	Trips per tour	Legs per trip	Trip distance in km (std)	Trip duration (std)	Tracking quality (std)
Green Class 1	4.32	2.73	1.92	27.4 (478.7)	0.52 (0.73)	0.85 (0.17)
Green Class 2	3.80	2.66	2.09	33.7 (568.2)	0.51 (0.75)	0.75 (0.24)
Yumuv	3.13	2.11	2.51	16.9 (100.4)	0.68 (0.91)	0.77 (0.23)
Geolife	1.70	2.37	1.00	36.1 (3163.5)	0.64 (0.94)	0.4 (0.32)

Another key part of tracking data analysis regards the temporal tracking quality of a dataset. Here, temporal tracking quality is defined as the temporal coverage of the tracking data (i.e., the completeness) and is computed with the Trackintel function `temporal_tracking_quality()` as explained in section 3.5.4. The results are given in the last column of Table 4. The three GNSS-based studies show a high coverage of more than 75% on average per user, whereas Geolife data only covers about 40% of the time on average per user. Figure 7 shows the distribution of the tracking quality over users. In the Geolife dataset, the temporal tracking quality largely differs across individuals. In comparison, the large majority of Green Class 1 participants reached a coverage of more then 0.7. The large difference between Geolife and the other datasets can be explained by the different hardware that was used in the studies. While the Geolife individuals were equipped with dedicated GPS-only trackers that are prone to localization problems when indoors or in urban canyons, the participants in the Green Class and yumuv studies were tracked with an app on their smart phone that uses the location API of the operating system. The latter has access to all GNSS systems in addition to GPS and can fall back to other technologies such as WIFI or cell tower triangulation if no satellite is available.

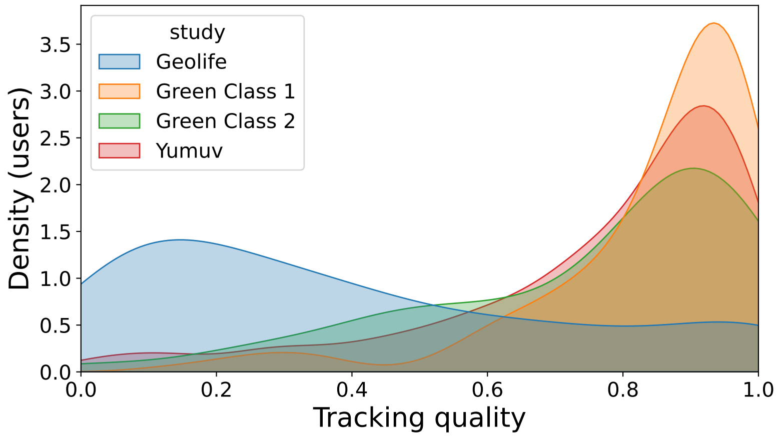
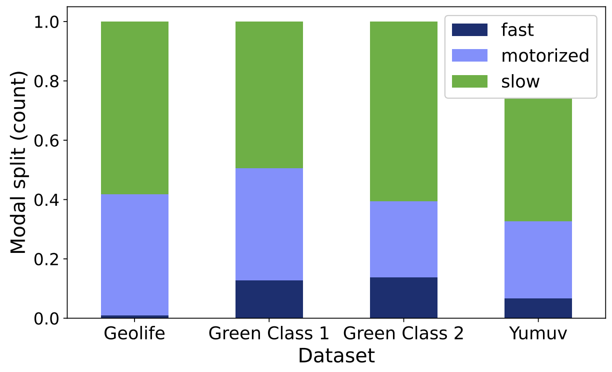
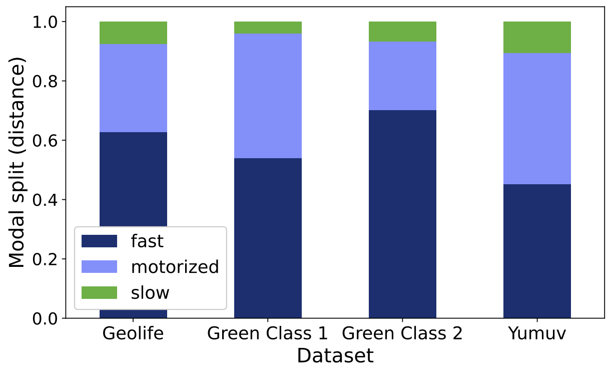


Figure 7: Distribution of the individual temporal tracking quality for the considered datasets.

We further compare the modal split of the tracking studies. The split is computed first as the number of triplegs per mode and secondly as the covered distance per mode. We use the Trackintel function `predict_transport_mode()` to approximate the modes for the Geolife dataset, since the original mode labels are not available for all participants and not all the time. In all other studies, high-quality mode labels are provided, and we aggregate them into the simplified categories of slow mobility (walk, bicycle, scooter), motorized mobility (tram, bus, car and motorbike) and fast mobility (airplane and train). The results are shown in Figure 8. The datasets differ significantly with respect to their modal split, which can be explained by the study target group, for example, Green Class participants were given full access to all public transport in Switzerland and are thus more likely to use trains (fast transport). Yumuv individuals on the other hand mostly live in urban areas and they were using the yumuv bundle of shared e- bicycles and scooters, which explains the higher proportion of slow mobility for yumuv.



(a) Split by count

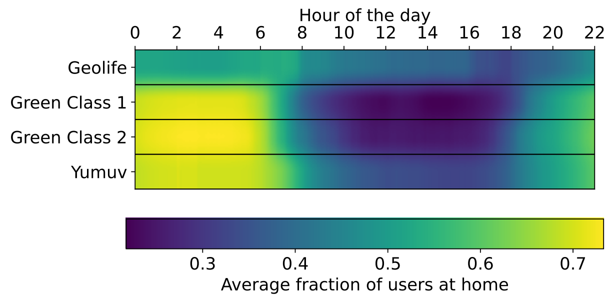


(b) Split by distance

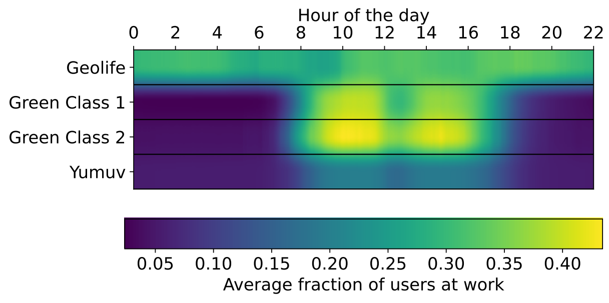
Figure 8: Comparison of modal split between datasets. The users of different studies differ considerably in terms of their usage of slow, motorized or fast transport.

Finally, we analyze the daily activity patterns of individuals. Specifically, the time periods where the individuals are at home and at work are computed. For the Green Class 1 & 2 studies, the activity-label for each staypoint is provided by the participants. For the Geolife and yumuv datasets, on the other hand, we adopt the Trackintel `location_identifier()` function that implements the OSNA algorithm [16] to infer the home and work locations. In Figure 9, the distribution of home and work staypoints over the course of a day is shown. Specifically, the average fraction of users with a staypoint labeled home (or work respectively) is shown for every minute of the day. The fraction of users at home (work) is thereby computed as the number of staypoints per day divided by the number of actively tracked users, where a user is actively tracked if there is at least one staypoint on that day. The working time between 8am and 5pm as well as the lunch breaks are clearly visible in Figure 9b for Green Class 1 & 2 and yumuv, although there are fewer work-staypoints for yumuv. While the home location is reliably identified for both yumuv and Geolife, the identification of the work location seems impaired for the Geolife dataset. As the OSNA algorithm simply selects the second-most visited location as work if the “home” and “work” labels overlap, the low tracking quality of the Geolife dataset (see Figure 7) could have affected the accuracy of the identification.

In summary, our study demonstrates the ease of comparing data from different sources on all levels of the movement data model and concerning various labels for the movement data. The standardized preprocessing functions implemented in Trackintel also help compare methods and explain possible discrepancies in the analysis results from the different datasets.



(a) Home-labeled staypoints



(b) Work-labeled staypoints

Figure 9: Distribution of activities over time.

5 Discussion and Conclusion

Quantitative analysis of human mobility currently suffers from a lack of a common data model for movement data and a standardization of preprocessing steps, limiting the reproducibility and comparability of scientific studies. This article presented Trackintel, a new open-source tool to address these problems. Trackintel implements a widely accepted conceptual data model for movement data and provides functionalities for the full life-cycle of human mobility data analysis: import and export of tracking data collected through various methods, preprocessing, data quality assessment, semantic enrichment, quantitative analysis and mining tasks, and visualization of data and results.

A particular strength of Trackintel is that it greatly simplifies the joint analysis of several movement datasets with different properties. This was shown in a case study where four different datasets where jointly preprocessed and analyzed. We used the analysis methods implemented in Trackintel to compare the datasets with respect to their trip properties, their tracking quality, their modal split and their daily activity patterns. It was demonstrated that rich insights about the characteristics of different tracking datasets can be easily obtained in Trackintel with few lines of code.

On the other hand, the compatibility of Trackintel with diverse datasets limits the capabilities of the analysis model. A good example is the transport mode prediction function provided by Trackintel that is based on a simple heuristic. A more sophisticated and powerful method can in principle be implemented for a specific dataset, however, the applicability of this method to other datasets will be limited by the availability of specific input data or additional context data.

Importantly, the purpose of Trackintel is not to provide a comprehensive set of analysis functions, but rather a high-quality implementation of standard aggregation and semantics-enrichment steps that are relevant for most tracking studies. Nevertheless, Trackintel will be continuously extended to incorporate the latest processing and analysis algorithms and to offer a wider variety of options for the preprocessing, analysis and visualization of movement data.

Finally, Trackintel does not aim to cover all preprocessing and analysis needs for every movement data study. However, due to the compatibility with Pandas and Geopandas, Trackintel can easily be integrated in a larger workflow that comprises a variety of Python data and spatial analysis libraries. In particular, it is targeted at providing the same reliability as these standard libraries. This is achieved through a strong compliance with Python library standards, including a high coverage of unit tests with both real and synthetic data, code reviews as quality checks and continuous integration pipelines. In this setup, new algorithms can be contributed easily without risking to break existing functionality. We therefore believe Trackintel can serve as a standard and well-trusted mobility processing tool.

6 Acknowledgement

This work was supported by the Swiss Data Science Center [C17-14] and the ETH Zurich Foundation [MI-01-19]; Additionally we would like to thank Christof Leutenegger, Sven Ruf, and Nishant Kumar for their code contributions to Trackintel, David Jonietz for helping to create the idea of Trackintel, and René Buffat and Jiří Kunčar for their technical input in the early stage of this project.

References

[1] Rein Ahas, Anto Aasa, Y Yuan, Martin Raubal, Zbigniew Smoreda, Yu Liu, Cezary Ziemlicki, Margus Tiru, and Matthew Zook. Everyday space–time geographies: using mobile phone-based sensor data to monitor urban activity in Harbin, Paris, and Tallinn. *International Journal of Geographical Information Science*, 29(11):2017–2039, 2015.

[2] Laura Alessandretti, Piotr Sapiezynski, Vedran Sekara, Sune Lehmann, and Andrea Baronchelli. Evidence for a conserved quantity in human mobility. *Nature Human Behaviour*, 2(7):485–491, July 2018. doi: 10.1038/s41562-018-0364-x.

[3] Ulf Aslak and Laura Alessandretti. Infostop: Scalable stop-location detection in multi-user mobility data. *arXiv preprint arXiv:2003.14370*, 2020.

[4] Kay W. Axhausen. Definition of movement and activity for transport modelling. In *Handbook of transport modelling*. Emerald Group Publishing Limited, 2007.

[5] Danya Bachir, Ghazaleh Khodabandelou, Vincent Gauthier, Mounim El Yacoubi, and Eric Vachon. Combining bayesian inference and clustering for transport mode detection from sparse and noisy geolocation data. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 569–584. Springer, 2018.

[6] Aleix Bassolas, Hugo Barbosa-Filho, Brian Dickinson, Xerxes Dotiwalla, Paul Eastham, Riccardo Gallotti, Gourab Ghoshal, Bryant Gipson, Surendra A. Hazarie, Henry Kautz, Onur Kucuktunc, Allison Lieber, Adam Sadilek, and José J. Ramasco. Hierarchical organization of urban mobility and its connection with city livability. *Nature Communications*, 10(1):4817, 2019. doi: 10.1038/s41467-019-12809-y.

[7] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[8] Dirk Brockmann, Lars Hufnagel, and Theo Geisel. The scaling laws of human travel. *Nature*, 439(7075):462–465, January 2006. doi: 10.1038/nature04292.

[9] Dominik Bucher, Francesca Mangili, Francesca Cellina, Claudio Bonesana, David Jonietz, and Martin Raubal. From location tracking to personalized eco-feedback: A framework for geographic information collection, processing and visualization to promote sustainable mobility behaviors. *Travel behaviour and society*, 14:43–56, 2019.

[10] Clément Calenge. The package "adehabitat" for the r software: a tool for the analysis of space and habitat use by animals. *Ecological modelling*, 197(3-4):516–519, 2006.

[11] Clement Calenge. Analysis of animal movements in r: the adehabitatlt package. *R Foundation for Statistical Computing, Vienna*, 2011.

[12] Serina Chang, Emma Pierson, Pang Wei Koh, Jaline Gerardin, Beth Redbird, David Grusky, and Jure Leskovec. Mobility network models of COVID-19 explain inequities and inform reopening. *Nature*, 589(7840):82–87, January 2021. doi: 10.1038/s41586-020-2923-3.

[13] Cynthia Chen, Jingtao Ma, Yusak Susilo, Yu Liu, and Menglin Wang. The promises of big data and small data for travel behavior (aka human mobility) analysis. *Transportation research part C: emerging technologies*, 68: 285–299, 2016.

[14] Qingqing Chen and Ate Poorthuis. Identifying home locations in human mobility data: an open-source R package for comparison and reproducibility. *International Journal of Geographical Information Science*, 35(7):1425–1448, July 2021. doi: 10.1080/13658816.2021.1887489.

[15] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10(2):112–122, 1973.

[16] Hariton Efstathiades, Demetris Antoniades, George Pallis, and Marios D Dikaiakos. Identification of key locations based on online social network activity. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 218–225. IEEE, 2015.

[17] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. DeepMove: Predicting human mobility with attentional recurrent networks. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, pages 1459–1468. ACM Press, 2018. ISBN 978-1-4503-5639-8. doi: 10.1145/3178876.3186058.

[18] Hannah Frick and Ioannis Kosmidis. tracker: Infrastructure for running and cycling data from gps-enabled tracking devices in r. *Journal of Statistical Software*, 82:1–29, 2017.

[19] Sean Gillies. The shapely user manual. 2013. URL https://shapely.readthedocs.io/en/stable/manual.html.

[20] Marta C. González, César A. Hidalgo, and Albert-László Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, June 2008. doi: 10.1038/nature06958.

[21] Anita Graser. MovingPandas: Efficient Structures for Movement Data in Python. *GI_Forum*, 1:54–68, 2019. doi: 10.1553/giscience2019_01_s54.

[22] Anita Graser. Tools for the analysis of movement data, 2020. URL https://github.com/anitagraser/movement-analysis-tools.

[23] Salman Haidri, Yaksh J Haranwala, Vania Bogorny, Chiara Renso, Vinicius Prado da Fonseca, and Amilcar Soares. Ptrail–a python package for parallel trajectory data preprocessing. *arXiv preprint arXiv:2108.13202*, 2021.

[24] Ramaswamy Hariharan and Kentaro Toyama. Project lachesis: parsing and modeling location histories. In *International Conference on Geographic Information Science*, pages 106–124. Springer, 2004.

[25] Ye Hong, Yanan Xin, Henry Martin, Dominik Bucher, and Martin Raubal. A clustering-based framework for individual travel behaviour change detection. In *11th International Conference on Geographic Information Science (GIScience 2021)-Part II*, 2021.

[26] Haosheng Huang, Georg Gartner, Jukka M. Krisp, Martin Raubal, and Nico Van de Weghe. Location based services: ongoing evolution and research agenda. *Journal of Location Based Services*, 12(2):63–93, April 2018. doi: 10/ghx2v9.

[27] Haosheng Huang, Yi Cheng, and Robert Weibel. Transport mode detection based on mobile phone network data: A systematic review. *Transportation Research Part C: Emerging Technologies*, 101:297–312, 2019.

[28] David Jonietz and Dominik Bucher. Continuous trajectory pattern mining for mobility behaviour change detection. In *LBS 2018: 14th International Conference on Location Based Services*, pages 211–230. Springer, 2018.

[29] Rocío Joo, Matthew E. Boone, Thomas A. Clay, Samantha C. Patrick, Susana Clusella-Trullas, and Mathieu Basille. Navigating through the R packages for movement. *Journal of Animal Ecology*, 89(1):248–267, 2020. doi: 10.1111/1365-2656.13116.

[30] Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, James McBride, Jacob Wasserman, Adrian Garcia Badaracco, Jeffrey Gerard, Alan D. Snow, Jeff Tratner, Matthew Perry, Carson Farmer, Geir Arne Hjelle, Micah Cochran, Sean Gillies, Lucas Culbertson, Matt Bartos, Brendan Ward, Giacomo Caria, Mike Taves, Nick Eubank, sangarshanan, John Flavin, Matt Richards, Sergio Rey, maxalbert, Aleksey Bilogur, Christopher Ren, Dani Arribas-Bel, Daniel Mesejo-León, and Leah Wasser. geopandas/geopandas: v0.10.2. October 2021. doi: 10.5281/zenodo.5573592.

[31] Carsten Keßler and Grant McKenzie. A geoprivacy manifesto. *Transactions in GIS*, 22(1):3–19, 2018. doi: 10.1111/tgis.12305.

[32] Jinsoo Kim, Jae Hun Kim, and Gunwoo Lee. GPS data-based mobility mode inference model using long-term recurrent convolutional networks. *Transportation Research Part C: Emerging Technologies*, 135:103523, February 2022. doi: 10.1016/j.trc.2021.103523.

[33] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–10, 2008.

[34] Robin Lovelace and Richard Ellison. stplanr: A package for transport planning. *The R Journal*, 10(2):7–23, 2018.

[35] Massimiliano Luca, Gianni Barlacchi, Bruno Lepri, and Luca Pappalardo. A survey on deep learning for human mobility. *ACM Computing Surveys*, 55(1):7:1–7:44, 2021. doi: 10.1145/3485125.

[36] Henry Martin, Henrik Becker, Dominik Bucher, David Jonietz, Martin Raubal, and Kay W. Axhausen. Begleitstudie SBB Green Class - Abschlussbericht. *Working Paper No. 1439, Institute for Transport Planning and Systems, ETH Zürich*, 2019. doi: 10.3929/ethz-b-000353337.

[37] Henry Martin, Daniel J. Reck, Kay W. Axhausen, and Martin Raubal. ETH Mobility Initiative Project MI-01-19 Empirical use and Impact analysis of MaaS. Technical report, ETH Zurich, 2021.

[38] Wes McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[39] Esteban Moro, Dan Calacci, Xiaowen Dong, and Alex Pentland. Mobility patterns are associated with experienced income segregation in large US cities. *Nature Communications*, 12(1):4633, December 2021. doi: 10.1038/s41467-021-24899-8.

[40] The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

[41] Luca Pappalardo, Filippo Simini, Salvatore Rinzivillo, Dino Pedreschi, Fosca Giannotti, and Albert-László Barabási. Returners and explorers dichotomy in human mobility. *Nature Communications*, 6(1):8166, November 2015. doi: 10.1038/ncomms9166.

[42] Luca Pappalardo, Filippo Simini, Gianni Barlacchi, and Roberto Pellungrini. scikit-mobility: a python library for the analysis, generation and risk assessment of mobility data, 2019.

[43] Adrian C. Prelipcean, Gyözö Gidófalvi, and Yusak O. Susilo. Transportation mode detection–an in-depth review of applicability and reliability. *Transport reviews*, 37(4):442–464, 2017.

[44] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seong Joon Kim, and Song Chong. On the Levy-Walk Nature of Human Mobility. *IEEE/ACM Transactions on Networking*, 19(3):630–643, June 2011. doi: 10.1109/TNET.2011.2120618.

[45] Angela Rout, Sophie Nitoslawski, Andrew Ladle, and Paul Galpern. Using smartphone-gps data to understand pedestrian-scale behavior in urban settings: A review of themes and approaches. *Computers, Environment and Urban Systems*, 90:101705, 2021.

[46] Arina de Jesus Amador Monteiro Sanches. Uma arquitetura e implementação do módulo de pré-processamento para biblioteca pymove. 2019.

[47] Christian M. Schneider, Vitaly Belik, Thomas Couronné, Zbigniew Smoreda, and Marta C. González. Unravelling daily human mobility motifs. *Journal of The Royal Society Interface*, 10(84):20130246, July 2013. doi: 10.1098/rsif.2013.0246.

[48] Stefan Schönfelder and Kay W. Axhausen. *Urban rhythms and travel behaviour: spatial and temporal phenomena of daily travel*. Routledge, 2016.

[49] Justin Shenk, Wolf Byttner, Saranraj Nambusubramaniyan, and Alexander Zoeller. Traja: A python toolbox for animal trajectory analysis. *Journal of Open Source Software*, 6(63):3202, 2021.

[50] Kamil Smolak, Katarzyna Siła-Nowicka, Jean-Charles Delvenne, Michał Wierzbiński, and Witold Rohm. The impact of human mobility data scales and processing on movement predictability. *Scientific Reports*, 11(1):1–10, 2021.

[51] Adir Solomon, Amit Livne, Gilad Katz, Bracha Shapira, and Lior Rokach. Analyzing movement predictability using human attributes and behavioral patterns. *Computers, Environment and Urban Systems*, 87:101596, 2021. doi: 10.1016/j.compenvurbsys.2021.101596.

[52] Chaoming Song, Tal Koren, Pu Wang, and Albert-László Barabási. Modelling the scaling properties of human mobility. *Nature Physics*, 6(10):818–823, October 2010. doi: 10.1038/nphys1760.

[53] Chaoming Song, Z. Qu, Nicholas Blumm, and Albert-László Barabasi. Limits of Predictability in Human Mobility. *Science*, 327(5968):1018–1021, February 2010. doi: 10.1126/science.1177170.

[54] Eran Toch, Boaz Lerner, Eyal Ben-Zion, and Irad Ben-Gal. Analyzing large-scale human mobility data: a survey of machine learning methods and applications. *Knowledge and Information Systems*, March 2018. doi: 10.1007/s10115-018-1186-x.

[55] Jorim Urner, Dominik Bucher, Jing Yang, and David Jonietz. Assessing the influence of spatio-temporal context for next place prediction using different machine learning approaches. *ISPRS International Journal of Geo-Information*, page 24, 2018.

[56] Peter Widhalm, Philippe Nitsche, and Norbert Brändie. Transport mode detection with realistic smartphone sensor data. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 573–576. IEEE, 2012.

[57] Andrew T Wilson. Tracktable trajectory analysis. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2014.

[58] Yanyan Xu, Serdar Çolak, Emre C. Kara, Scott J. Moura, and Marta C. González. Planning for electric vehicle needs by coupling charging profiles with urban mobility. *Nature Energy*, 3(6):484–493, June 2018. doi: 10.1038/s41560-018-0136-x.

[59] Qing Yu and Jian Yuan. Transbigdata: A python package for transportation spatio-temporal big data processing, analysis and visualization. *Journal of Open Source Software*, 7(71):4021, 2022.

[60] Pengxiang Zhao, David Jonietz, and Martin Raubal. Applying frequent-pattern mining and time geography to impute gaps in smartphone-based human-movement data. *International Journal of Geographical Information Science*, pages 1–29, January 2021. doi: 10.1080/13658816.2020.1862126.

[61] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):1–41, 2015.

[62] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800, 2009.

[63] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.

[64] Esteban Zimányi, Mahmoud Sakr, and Arthur Lesuisse. Mobilitydb: A mobility database based on postgresql and postgis. *ACM Transactions on Database Systems (TODS)*, 45(4):1–42, 2020.

A Documentation score

A.1 Python

The documentation score reported in Table 1 for python libraries is based on the pyOpenSci package peer-review evaluation critera[15]

- Has an Open Software Initiative (OSI) approved license.
- Contains a README with instructions for installing the development version.
- Contains a vignette (notebook) with examples of its essential functions and uses.
- Has a test suite.
- Has continuous integration, such as Travis CI, AppVeyor, CircleCI, and/or others.
- Includes documentation with examples for all functions.

A.2 R

The documentation score reported in Table 1 for R libraries is based on the ROpenScie package peer-review evaluation critera[16]

- Does the package have a CRAN accepted license?
- The package contains a reasonably complete readme with devtools install instructions.
- The package contains a vignette with examples of its essential functions.
- The package contains unit tests.
- The repository has continuous integration with Travis and/or another service.
- Package available on CRAN?

[15] https://www.pyopensci.org/contributing-guide/intro.html
[16] https://ropensci.org/