itinerum-tripkit Documentation

Release 0.0.14

Kyle Fitzsimmons

Jun 22, 2022

Contents

1 Installation **3**
- 1.1 Virtual Environments . . . 3
- 1.2 Dependencies . . . 4

2 Configuration **7**
- 2.1 Generating the Configuration . . . 7

3 Quick Start **9**
- 3.1 Load Data . . . 9
- 3.2 Run Trip Detection on a User . . . 9
- 3.3 Run Complete Days Summaries on a User . . . 10
- 3.4 Run Semantic Location Activity Detection on a User . . . 10
- 3.5 Run OSRM Map Matching on a Trip . . . 10

4 API **13**
- 4.1 CSV Parser . . . 14
- 4.2 I/O . . . 15
- 4.3 Database . . . 18

5 Library Objects **21**
- 5.1 User . . . 21
- 5.2 Trip . . . 21
- 5.3 Trip Point . . . 22
- 5.4 Day Summary . . . 22

6 Outputs **23**
- 6.1 Trips . . . 23
- 6.2 Complete Days . . . 23

7 FAQ **25**

8 Contributing **27**
- 8.1 Style Guide . . . 27

9 Indices and tables **29**

Python Module Index **31**

Index 33

The itinerum-tripkit is a processing library for handling exports from the Itinerum data collection platform. The tripkit provides methods for handling and cleaning GPS .csv data, processing collected points to trip features, and writing GIS-compatible files for visualizing results.

The *Quick Start* shows the most common usage patterns of the tripkit library while the *API* section will detail how to interact with the library's core methods directly.

CHAPTER 1

Installation

The **itinerum-tripkit** can be installed as a library using pip can be included in your own project by cloning: http://github.com/TRIP-Lab/itinerum-tripkit and copying the `tripkit` directory as a submodule.

1.1 Virtual Environments

It is recommended to use venv to keep the tripkit dependency versions isolated from system packages.

1.1.1 Linux & MacOS

```
$ python3 -m venv tripkit-venv
$ source ./tripkit-ven/bin/activate
```

1.1.2 Windows

```
PS C:\Code\itinerum-tripkit> python -m venv tripkit-venv
PS C:\Code\itinerum-tripkit> .\tripkit-venv\Scripts\Activate.ps1
```

PowerShell Note: With PowerShell, `Set-ExecutionPolicy Unrestricted -Force` may be required to allow the `Activate.ps1` script to run. The PowerShell prompt must then be restarted for these permissions to take effect.

1.2 Dependencies

1.2.1 Windows

Windows does not provide a build environment by default with libraries relied upon by the GDAL package. Instead, the Visual C++ Redistributable for Visual Studio 2015 (13.4MB) can be installed to provide the necessary system libraries.

If an existing GDAL installation is available to Python, it may be possible to skip this step.

Compiled Python Packages

On Windows without a C/C++ build environment, some packages requiring will fail to install using *pip*. Instead, compiled wheel versions can be downloaded from various mirrors and installed with pip directly from file.

```
(tripkit-venv) PS C:\Code\itinerum-tripkit> pip install .\Fiona-1.8.6-cp37-cp37m-win_
↪amd64.whl
```

Compiled packages to install:

- GDAL: https://www.lfd.uci.edu/~gohlke/pythonlibs/#gdal
- Fiona: https://www.lfd.uci.edu/~gohlke/pythonlibs/#fiona

GDAL (Alternative)

If the above is not successful, another method is using the gisinternals.com pre-compiled binaries. For your system version (likely *MSVC 2017 / x64*), click "Downloads". From the downloads page, the core GDAL library is all that is needed (*gdal-204-1911-64-core.msi*).

Install this file and set two Windows environment variables:

- Append to PATH: `C:\Program Files\GDAL`
- Create GDAL_DATA: `C:\Program Files\GDAL\gdal-data`

After setting these variables, close and re-open the command prompt (re-activate the virtual environment if using) and the Python dependencies can be installed.

First the GDAL library must be installed for geospatial operations and outputs.

Optional Components

Scikit-learn

Scikit-learn can optionally be installed for optimized clustering such as by swapping out the included K-Means implementation:

- Scikit-learn: https://www.lfd.uci.edu/~gohlke/pythonlibs/#scikit-learn

OSRM

The **itinerum-tripkit** provides interfaces for submitting map matching queries to an OSRM API and writing results to file.

The instructions that follow use the Multi-Level Djikstra processing pipelines recommended by Project OSRM.

Installing the OSRM API with Docker containers

1. Download an OSM extract for your region (ex. Québec)

```
$ mkdir osrm && cd osrm
$ wget http://download.geofabrik.de/north-america/canada/quebec-latest.osm.pbf
```

2. Process the OSM data using the default network profiles included with OSRM:

```
# car
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-extract -p /opt/car.lua /data/
↪quebec-latest.osm.pbf
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-partition /data/quebec-latest
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-customize /data/quebec-latest
$ mkdir car
$ mv quebec-latest.orsm* car

# bike
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-extract -p /opt/bicycle.lua /
↪data/quebec-latest.osm.pbf
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-partition /data/quebec-latest
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-customize /data/quebec-latest
$ mkdir bicycle
$ mv quebec-latest.orsm* bicycle

# walking
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-extract -p /opt/foot.lua /data/
↪quebec-latest.osm.pbf
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-partition /data/quebec-latest
$ docker run -t -v $(pwd):/data osrm/osrm-backend osrm-customize /data/quebec-latest
$ mkdir foot
$ mv quebec-latest.orsm* foot
```

3. Run the Docker OSRM API containers on ports `5000-5002` to reverse proxy for public access

```
$ docker run -d --restart always -p 5000:5000 -v $(pwd)/car:/data osrm/osrm-backend␣
↪osrm-routed --algorithm MLD --max-matching-size=5000 /data/quebec-latest.osrm

$ docker run -d --restart always -p 5001:5000 -v $(pwd)/bicycle:/data osrm/osrm-
↪backend osrm-routed --algorithm MLD --max-matching-size=5000 /data/quebec-latest.
↪osrm

$ docker run -d --restart always -p 5002:5000 -v $(pwd)/foot:/data osrm/osrm-backend␣
↪osrm-routed --algorithm MLD --max-matching-size=5000 /data/quebec-latest.osrm
```

CHAPTER 2

Configuration

The **itinerum-tripkit** is configured by a global configuration object that is passed to the class at initialization. This can be created either as a Python file of global variables that is imported or defined as a bare class Config object and named `tripkit_config`.

2.1 Generating the Configuration

The following parameters are accepted by **itinerum-tripkit**:

`DATABASE_FN`	The filename to be used for the cache SQLite database.
`INPUT_DATA_DIR`	Directory of the unpacked TripKit export .csv files. Usually a subdirectory of the `./input` directory.
`INPUT_DATA_TYPE`	Data source: `itinerum` or `qstarz`
`OUTPUT_DATA_DIR`	Output directory to save processed export data.
`SUBWAY_STATIONS_FP`	Relative filepath of subway .csv data for connecting gaps during trip detection algorithms.
`TRIP_DETECTION_BREAK_INTERVAL_SECONDS`	Minimum stop time for breaking GPS coordinates into trips.
`TRIP_DETECTION_SUBWAY_BUFFER_METERS`	Buffer in meters for associating a trip end with a subway station entrance.
`TRIP_DETECTION_COLD_START_DISTANCE_METERS`	Leeway distance in meters for allowing a device acquire a GPS fix before inferring that an intemediary trip with missing data has occured.
`TRIP_DETECTION_ACCURACY_CUTOFF_METERS`	Minimum horizontal accuracy in meters for including GSP points within trip detection algorithms. Greater values indicate worse accuracy; generally 30-50 is deemed an acceptable range.

Process parameters

These parameters are only needed if their related processes will be run.

TIMEZONE	The timezone name as described within the tzdata database for complete days detection (e.g., America/Montreal)
SEMANTIC_LOCATIONS	Mapping of semantic locations to *latitude longitude* columns within survey responses (see below for example).
SEMANTIC_LOCATION_PROXIMITY_METERS	Buffer distance in meters to consider a GPS point to be at a semantic location.

Extra parameters

These parameters are to configure plug-in processes (OSRM map matching API below). Check the plug-in source code to see what is expected in these cases.

MAP_MATCHING_BIKING_API_URL	Endpoint for OSRM bicycle network map maptching.
MAP_MATCHING_DRIVING_API_URL	Endpoint for OSRM car network map maptching.
MAP_MATCHING_WALKING_API_URL	Endpoint for OSRM foot network map maptching.

CHAPTER 3

Quick Start

The **itinerum-tripkit** library provides simple interfaces to load *.csv* data into a local SQLite database and providing Python `class` objects (such as *tripkit.models.User* and *tripkit.models.Trip*) to represent this data for processing and inferring metadata.

3.1 Load Data

To initilize the libary, a configuration object is expected (see *Generating the Configuration*).

First, input .csv data be loaded to the **itinerum-tripkit** cache database:

```
from tripkit import TripKit
import tripkit_config

tripkit = TripKit(config=tripkit_config)
tripkit.setup()
```

After data has been loaded to the database, survey participants or *users* can be loaded as a list of *tripkit.models.User* objects:

```
users = tripkit.load_users()
for user in users:
    print(len(user.coordinates))
```

Note: On first run, .csv data will be imported if the table `user_survey_responses` *does not exist in the cache database. To delete the cache, the temporary files can be deleted between runs.*

3.2 Run Trip Detection on a User

Instead of running trip detection on the whole survey, it is possible to focus on a single user in detail. For writing new processing libraries, this is often an essential first step.

```
user = tripkit.load_users(uuid='00000000-0000-0000-0000-000000000000')
params = {
    'load_subway_entrances': tripkit.database.load_subway_entrances(),
    'break_interval_seconds': tripkit_config.TRIP_DETECTION_BREAK_INTERVAL_SECONDS,
    'subway_buffer_meters': tripkit_config.TRIP_DETECTION_SUBWAY_BUFFER_METERS,
    'cold_start_distance': tripkit_config.TRIP_DETECTION_COLD_START_DISTANCE_METERS,
    'accuracy_cutoff_meters': tripkit_config.TRIP_DETECTION_ACCURACY_CUTOFF_METERS
}
trips = tripkit.process.trip_detection.triplab.algorithm.run(user.coordinates,
                                                             parameters=params)
```

3.3 Run Complete Days Summaries on a User

When trips have been detected for a user, the complete days summary process can be run on individual users. This will check to see if a day is "complete" (contains no missing trips), "incomplete", or "inactive". There are additional rules to consider days as "complete" if there is an inactive day between two complete days; it is recommended to review the process source code.

```
user = tripkit.load_users(uuid='00000000-0000-0000-0000-000000000000')
trip_day_summaries = tripkit.process.complete_days.triplab.counter.run(user.trips,
                                                                       tripkit_
↪config.TIMEZONE)
tripkit.database.save_trip_day_summaries(user, trip_day_summaries, tripkit_config.
↪TIMEZONE)
```

3.4 Run Semantic Location Activity Detection on a User

If common user locations are available within survey responses or supplied separately (such as from the outputs of a clustering process), dwell times from nearby GPS points can be tallied. Note: the `Coordinate` model is currently created on-the-fly as demonstrated, but this should soon be available as an included library class-object.

```
Coordinate = namedtuple('Coordinate', ['latitude', 'longitude'])
user = tripkit.load_users(uuid='00000000-0000-0000-0000-000000000000')
locations = {
    'home': Coordinate(latitude=45.5, longitude=-73.5)
}
tripkit.io.write_activity_locations_geojson(tripkit_config, fn_base=user.uuid,␣
↪locations=locations)
summary = tripkit.process.activities.triplab.detect.run(
    user, locations, proximity_m=tripkit_config.ACTIVITY_LOCATION_PROXIMITY_METERS,␣
↪timezone=tripkit_config.TIMEZONE)
dwell_time_summaries = [summary]  # usually, multiple users would be summarized for␣
↪output
tripkit.io.write_user_summaries_csv(tripkit_config, dwell_time_summaries)
```

3.5 Run OSRM Map Matching on a Trip

If an OSRM server is available, map matching queries can be passed to the API and the response saved to a GIS-friendly format (*.geojson* or *.gpkg*). The API query is limited by URL length, so map matching should be done for a single trip and especially long trips may have to be supplied in chunks.

```
user = tripkit.load_users(uuid='00807c5b-7542-4868-8462-14b79a9fcc9f',
                          start=datetime(2017, 11, 29),
                          end=datetime(2017, 11, 30))
map_matcher = tripkit.process.map_match.osrm(tripkit_config)
mapmatched_results = map_matcher.match(coordinates=user.coordinates, matcher='DRIVING
↪')
tripkit.io.write_mapmatched_geojson(cfg=tripkit_config, fn_base=user.uuid,␣
↪results=mapmatched_results)
```

CHAPTER 4

API

class tripkit.**TripKit**(*config*)

The base TripKit object provides an interface for working with .csv data exported from the Itinerum platform or QStarz GPS data loggers. The object is passed the *config* at initialization which should be an imported Python file of global variables or class with the same attributes as expected here.

This TripKit object is the entry API for loading .csv data to an SQLite database (used as cache), running algorithms on the GPS data, and visualizing or exporting the results to GIS-friendly formats.

The TripKit instance is usually created in your main module like this:

```
from tripkit import TripKit
import tripkit_config

tripkit = TripKit(config=tripkit_config)
tripkit.setup()
```

Parameters **config** – An imported Python file of global variables or a bare config class with the same attributes, see *Generating the Configuration* for more information.

check_setup()

Raises exception if *UserSurveyResponse* table is not found in database.

csv

Provides access to the CSV parser objects.

database

Provides access to the cache database object.

io

Provides access to the file reading and writing functions.

load_user_by_orig_id(*orig_id*, *load_trips=True*, *start=None*, *end=None*)

Returns all available users as *tripkit.models.User* objects from the database

Parameters

- **orig_id** – An individual user's original ID from a non-Itinerum dataset to load
- **load_trips** (*boolean, optional*) – Supply False to disable automatic loading of trips to py:class:*tripkit.models.User* objects on initialization
- **start** (*datetime, optional*) – Miminum timestamp bounds (inclusive) for loading user coordinate and prompts data
- **end** (*datetime, optional*) – Maximum timestamp bounds (inclusive) for loading user coordinate and prompts data

Return type *tripkit.models.User*

load_users(*uuid=None*, *load_trips=True*, *load_locations=True*, *limit=None*, *start=None*, *end=None*)

Returns all available users as *tripkit.models.User* objects from the database

Parameters

- **uuid** (*string, optional*) – Supply an individual user's UUID to load
- **load_trips** (*boolean, optional*) – Supply False to disable automatic loading of trips to *tripkit.models.User* objects on initialization
- **load_locations** (*boolean, optional*) – Supple False to disable automatic loading of activity locations to *tripkit.models.User* objects on initialization
- **limit** (*integer, optional*) – Maximum number of users to load
- **start** (*datetime, optional*) – Mininum timestamp bounds (inclusive) for loading user coordinate and prompts data
- **end** (*datetime, optional*) – Maximum timestamp bounds (inclusive) for loading user coordinate and prompts data

Return type list of *tripkit.models.User*

process

Provides access to the GPS point and trip processing algorithm submodules.

setup(*force=False*, *generate_null_survey=False*)

Create the cache database tables if the `UserSurveyResponse` table does not exist.

Parameters

- **force** (*boolean, optional*) – Supply *True* to force creation of a new cache database
- **generate_null_survey** (*boolean, optional*) – Supply *True* to generate an empty survey responses table for coordinates-only data

4.1 CSV Parser

class tripkit.csvparser.**ItinerumCSVParser**(*database*)

Parses Itinerum platform csv files and loads to them to a cache database.

Parameters **database** – Open Peewee connection the cache database

generate_null_survey(*input_dir*)

Wrapper function to generate null survey responses for each user in coordinates.

Parameters **input_dir** – Directory containing input .csv data

load_export_cancelled_prompt_responses(*input_dir*)
Loads Itinerum cancelled prompt responses data to the cache database. For each .csv row, the data is fetched by column name if it exists and cast to appropriate types as set in the database.

Parameters **input_dir** – The directory containing the *self.cancelled_prompt_responses.csv* data file.

load_export_coordinates(*input_dir*)
Loads Itinerum coordinates data to the cache database.

Parameters **input_dir** – The directory containing the *self.coordinates_csv* data file.

load_export_prompt_responses(*input_dir*)
Loads Itinerum prompt responses data to the cache database. For each .csv row, the data is fetched by column name if it exists and cast to appropriate types as set in the database.

Parameters **input_dir** – The directory containing the *self.prompt_responses.csv* data file.

load_export_survey_responses(*input_dir*)
Loads Itinerum survey responses data to the cache database.

Parameters **input_dir** – The directory containing the *self.survey_responses_csv* data file.

load_trips(*trips_csv_fp*)
Loads trips processed by the web platform itself. This is mostly useful for comparing current algorithm results against the deployed platform's version.

Parameters **trips_csv_fp** – The full filepath of the downloaded trips *.csv* file for a survey.

class tripkit.csvparser.**QstarzCSVParser**(*config*, *database*)
Parses Qstarz csv files and loads to them to a cache database.

Parameters

- **config** –
- **database** – Open Peewee connection the cache database
- **csv_input_dir** – Path to the directory containing the input coordinates .csv data

generate_null_survey(*input_dir*)
Wrapper function to generate null survey responses for each user in coordinates.

Parameters **input_dir** – Directory containing input .csv data

load_export_coordinates(*input_dir*)
Loads QStarz coordinates data to the cache database.

Parameters **input_dir** – The directory containing the *self.coordinates_csv* data file.

load_user_locations(*input_dir*)
Loads QStarz user locations data to the cache database.

Parameters **input_dir** – The directory containing the *self.locations_csv* data file.

4.2 I/O

class tripkit.io.**IO**(*cfg*)

class tripkit.io.**CSVIO**(*cfg*)

write_activities_daily(*daily_summaries*, *extra_cols=None*, *append=False*)
Write the user activity summaries by date with a record for each day that a user participated in a survey.

Parameters

- **daily_summaries** (*list of dict*) – Iterable of user summaries for row records.
- **append** (*boolean, optional*) – Toggles whether summaries should be appended to an existing output file.

write_activity_summaries(*summaries*, *append=False*)
Write the activity summary data consisting of complete days and trips tallies with a record per each user for a survey.

Parameters

- **summaries** (*list of dict*) – Iterable of user summaries for row records
- **append** (*boolean, optional*) – Toggles whether summaries should be appended to an existing output file.

write_complete_days(*trip_day_summaries*, *append=False*)
Write complete day summaries to .csv with a record per day per user over the duration of their participation in a survey.

Parameters

- **trip_day_summaries** (*list of dict*) – Iterable of complete day summaries for each user enumerated by uuid and date.
- **append** (*boolean, optional*) – Toggles whether summaries should be appended to an existing output file.

write_condensed_activity_locations(*user*, *append=True*)
Write or append the provided user's activity locations to file.

Parameters

- **locations** (*list of dict*) – Iterable of user summaries for row records.
- **append** (*boolean, optional*) – Toggles whether summaries should be appended to an existing output file.

write_condensed_trip_summaries(*user*, *trip_summaries*, *complete_day_summaries*, *append=False*)
Write the trip summaries with added columns for labeled trip origins/destinations and whether a trip occured on a complete trip day.

Parameters

- **daily_summaries** (*list of dict*) – Iterable of user summaries for row records.
- **append** (*boolean, optional*) – Toggles whether summaries should be appended to an existing output file.

write_trip_summaries(*fn_base*, *summaries*, *extra_fields=None*, *append=False*)
Write detected trip summary data to csv consisting of a single record for each trip.

Parameters

- **fn_base** (*str*) – The base filename to prepend to the output csv file.
- **summaries** (*list of dict*) – Iterable of trip summaries for row records.
- **extra_fields** (*list, optional*) – Additional columns to append to csv (must have matching key in *summaries* object).

- **append** (*boolean, optional*) – Append data to an existing .csv file.

write_trips(*fn_base*, *trips*, *extra_fields=None*, *append=False*)
Write detected trips data to a csv file.

Parameters

- **fn_base** (*str*) – The base filename to prepend to the output csv file
- **trips** – Iterable of database trips to write to csv file
- **trips** – list of *tripkit.models.Trip*

class tripkit.io.**GeoJSONIO**(*cfg*)

write_activity_locations(*fn_base*, *locations*)
Write activity locations (from config or detected) to a geojson file.

Parameters

- **fn_base** (*str*) – The base filename to prepend to each output geojson file.
- **locations** (*dict*) – A dictionary object of a user's survey responses containing columns with activity location latitude and longitudes.

write_inputs(*fn_base*, *coordinates*, *prompts*, *cancelled_prompts*)
Writes input coordinates, prompts and cancelled prompts data selected from cache to individual geojson files.

Parameters

- **fn_base** (*str*) – The base filename to prepend to each output geojson file.
- **coordinates** (list of tripkit.database.Coordinate) – Iterable of database coordinates to write to geojson file.
- **prompts** (list of tripkit.database.PromptResponse) – Iterable of database prompts to write to geojson file.
- **cancelled_prompts** (list of tripkit.database.CancelledPromptResponse) – Iterable of database cancelled prompts to write to geojson file.

write_mapmatch(*fn_base*, *results*)
Writes map matching results from API query to geojson file.

Parameters

- **fn_base** (*str*) – The base filename to prepend to the output geojson file
- **results** – JSON results from map matching API query

write_trips(*fn_base*, *trips*)
Writes detected trips data selected from cache to geojson file.

Parameters

- **fn_base** (*str*) – The base filename to prepend to the output geojson file
- **trips** (list of *tripkit.models.Trip*) – Iterable of database trips to write to geojson file

class tripkit.io.**GeopackageIO**(*cfg*)

write_activity_locations(*fn_base*, *locations*)
Write activity locations (from config or detected) to a geopackage file.

Parameters

- **fn_base** (*str*) – The base filename to prepend to each output geopackage file.
- **locations** (*dict*) – A dictionary object of a user's survey responses containing columns with activity location latitude and longitudes.

write_inputs(*fn_base*, *coordinates*, *prompts*, *cancelled_prompts*)
Writes input coordinates, prompts and cancelled prompts data selected from cache to individual geopackage files.

Parameters

- **fn_base** (*str*) – The base filename to prepend to each output geopackage file.
- **coordinates** (list of `tripkit.database.Coordinate`) – Iterable of database coordinates to write to geopackage file.
- **prompts** (list of `tripkit.database.PromptResponse`) – Iterable of database prompts to write to geopackage file.
- **cancelled_prompts** (list of `tripkit.database.CancelledPromptResponse`) – Iterable of database cancelled prompts to write to geopackage file.

write_trips(*fn_base*, *trips*)
Writes detected trips data to a geopackage file.

Parameters

- **fn_base** – The base filename to prepend to the output geopackage file
- **trips** – Iterable of database trips to write to geopackage file
- **fn_base** – str
- **trips** – list of *tripkit.models.Trip*

4.3 Database

class tripkit.database.**Database**(*config*)
Handles itinerum-tripkit interactions with the cached database using peewee.

bulk_insert(*Model*, *rows*, *chunk_size=50000*)
Bulk insert an iterable of dictionaries into a supplied Peewee model by `chunk_size`.

Parameters

- **Model** – Peewee database model of target table for inserts.
- **rows** (*list*) – Iterable of dictionaries matching table model for bulk insert.
- **chunk_size** (*int, optional*) – Number of rows to insert per transaction.

clear_trips(*user=None*)
Clears the detected trip points table or for an individual user.

Parameters **user** – (Optional) Delete trips for particular user only.

count_users()
Returns a count of all survey responses in cache database.

create()
Creates all the tables necessary for the itinerum-tripkit cache database.

delete_user_from_table(*Model*, *user*)
Deletes a given user's records from a table in preparation for overwriting.

drop()
Drops all cache database tables.

get_uuid(*orig_id*)
Returns the database uuid for a user's original id from a non-Itinerum dataset.

Parameters **orig_id** – The original dataset's user id for an individual user.

load_activity_locations(*user*)
Queries cache database for activity locations known for a user.

Parameters **user** – A database user response record

load_subway_entrances()
Queries cache database for all available subway entrances.

load_trip_day_summaries(*user*)
Load the daily trip summaries for a given user as dict.

Parameters **user** – A database user response record with a populated *detected_trip_day_summaries* relation.

load_trips(*user*, *start=None*, *end=None*)
Load the sorted trips for a given user as list.

Parameters **user** – A database user response record with a populated *detected_trip_coordinates* relation.

load_user(*uuid*, *start=None*, *end=None*)
Loads user by `uuid` to an itinerum-tripkit `User` object.

Parameters

- **uuid** – A individual user's UUID from within an Itinerum survey.
- **start** – *Optional.* Naive datetime object (set within UTC) for selecting a user's coordinates start period.
- **end** – *Optional.* Naive datetime object (set within UTC) for selecting a user's coordinates end period.

save_trip_day_summaries(*user*, *trip_day_summaries*, *timezone*, *overwrite=True*)
Saves the daily summaries for detected trip days to cache database. This table with be recreated on each save by default.

Parameters

- **user** (*tripkit.models.User*) – A database user response record associated with the trip day summaries.
- **trip_day_summaries** (list of *tripkit.models.DaySummary*) – List of daily summaries from a daily trip counts algorithm.
- **timezone** (*str*) – The tz database timezone name for the location that was used to localize the complete days detection.
- **overwrite** (*boolean, optional*) – Provide *False* to keep existing summaries for user in database.

save_trips(*user*, *trips*, *overwrite=True*)

Saves detected trips from processing algorithms to cache database. This table will be recreated on each save by default.

Parameters

- **user** (*tripkit.models.User*) – A database user response record associated with the trip records.
- **trips** (list of *tripkit.models.Trip*) – Iterable of detected trips from a trip processing algorithm.

CHAPTER 5

Library Objects

5.1 User

The *tripkit.models.User* object provides the representation of a user in a survey with their associated coordinates, prompt, cancelled prompts and trips (if available in cache).

class tripkit.models.**User**(*db_user*)

Parameters **db_user** – The peewee survey response for a given user.

Variables

- **activity_locations** (*list*) – A user's saved or detected activity locations. This is loaded automatically when a User is initialized by *tripkit.database.Database.load_user()*.
- **trips** (*list*) – A user's detected trips. This is loaded automatically when a User is initialized by *tripkit.database.Database.load_user()*.

5.2 Trip

The *tripkit.models.Trip* object provides the representation of a user's trip with some inferred properties about it.

class tripkit.models.**Trip**(*num*, *trip_code*)

Parameters

- **num** (*int*) – The integer index (starting at 1) of all the detected trip events (completer or incomplete) as user has made over the duration of their survey participation.
- **trip_code** (*int*) – The integer code representing the detected trip type.

Variables **points** (list of *tripkit.models.TripPoint*) – The timestamp-ordered points that comprise this Trip.

5.3 Trip Point

The *tripkit.models.TripPoint* object provides the respresentation of a user's GPS coordinates after having been processed by a trip detection algorithm.

class tripkit.models.**TripPoint**(*database_id*, *latitude*, *longitude*, *h_accuracy*, *distance_before*, *trip_distance*, *period_before*, *timestamp_UTC*)

Parameters

- **database_id** (*integer*) – The GPS point's original database coordinates record id.
- **latitude** (*float*) – The GPS point's latitude.
- **longitude** (*float*) – The GPS point's longitude.
- **h_accuracy** (*float*) – The reported horizontal accuracy of a GPS point.
- **distance_before** (*float*) – The distance between this point and point immediately prior in meters.
- **trip_distance** (*float*) – The cumulative distance of the current trip so far in meters.
- **period_before** (*integer*) – The number of seconds passed since the last recorded point.
- **timestamp_UTC** (*datetime*) – The point's naive datetime localized to UTC.

Variables **timestamp_epoch** (*int*) – The point's datetime within the UNIX epoch format.

5.4 Day Summary

The *tripkit.models.DaySummary* object provides the representation of complete trip days after processing a user's trips by a complete days detection algorithm.

class tripkit.models.**DaySummary**(*timezone*, *date*, *has_trips*, *is_complete*, *start_point*, *end_point*, *consecutive_inactive_days*, *inactivity_streak*)

Parameters

- **timezone** (*str*) – String representation of the timezone as listed in the IANA tz database (e.g., America/Montreal). See: https://en.wikipedia.org/wiki/List_of_tz_database_time_zones
- **date** (*datetime.Date*) – The naive date used for determining a complete trip day.
- **has_trips** (*bool*) – Boolean value to indicate whether a date contains any trips.
- **is_complete** (*bool*) – Boolean value to indicate whether a date with trips is considered complete or not.
- **start_point** (*object*) – The database point for the first available GPS coordinate on a localized date.
- **end_point** (*object*) – The database point for the last available GPS coordinate on a localized date.
- **consecutive_inactive_days** (*int*) – The total number of days in the latest inactivity streak (reset on any complete day).
- **inactivity_streak** (*int*) – The longest streak of consecutively inactive days for a user.

CHAPTER 6

Outputs

6.1 Trips

6.2 Complete Days

The Complete Days/Trip Lab process iterates over the trips by 24-hour cycle (00:00:00-23:59:59) in the supplied timezone. For each day that a user participates in a study, the process checks if any trips are labeled as missing and if none exist, the day is labelled as "is_complete". If no trips exist, the day is not labeled as complete.

6.2.1 Special cases

- It is common that a user may legitimately make no trips on a participation day. In the case that there is a day without any trips, but the day prior and after are labeled complete and the distance between the last trip end and the next trip start is <= 750 meters, this day will be labeled "is_complete". Cases with more than 1 day without trips are not considered by this rule.

6.2.2 Notes

- The output results are grouped by `uuid` and ordered by `date`.
- Boolean values are represented as `1` (True) or `0` (False).

uuid	The unique user ID in the survey.
date	The date (localized by timezone) for complete day record.
has_trips	Boolean to indicate whether date contains any trips.
is_complete	Boolean to indicate if date contains trips and no missing trips.
start_latitude	The latitude of the date's first trip.
start_longitude	The longitude of the date's first trip.
end_latitude	The latitude of the date's last trip.
end_longitude	The longitude of the date's last trip.
consecutive_inactive_days	The current tally of inactive days in a row (reset each complete day).
inactivity_streak	The maximum tally of incomplete days in a row for a user.

CHAPTER 7

FAQ

Q: Why are there additional points with duplicate timestamps in my detected trips?

A: Trip detection creates a labeled, uninterrupted trajectory from a user's coordinates. Occasionally gaps occur between trips, either because of device handling of geofences, signal loss due to interference or battery drain, or a user temporarily pausing data collection. When this occurs, a missing trip will be generated.

If a missing trip is sufficiently small and immediately preceding a detected complete trip, it will be prepended to the complete trip. When this occurs, a new point is generated with the end location of the previous trip and the same timestamp as the start (now the second point) of the next complete trip.

CHAPTER 8

Contributing

8.1 Style Guide

8.1.1 Notes

The **itinerum-tripkit** is a modular library that is built to be extended. The most common place to contribute will be adding a new *process* (*itinerum-tripkit/tripkit/process*) for modifying or building new GPS detection algorithms. The process filepath format is: *tripkit/process/<descriptive-name>/<organization>/library_files.py*. Try to group additions as best as possible within existing directories if a one exists, but otherwise it is encouraged to be descriptive. If a suitable name doesn't exist, it's better to create a new directory altogether.

Database

Library readability is prioritized over absolute execution performance, although both are desired. To keep the Python code base neater, the Peewee ORM is used for database operations. Peewee is used instead of SQLAlchemy to help keep the library compact while providing comparable functionality. The database functions are intended to be SQL database agnostic, however, bulk inserts for initial data loading is tightly coupled to SQLite. Additional database engines will require specially handling data import from *.csv*.

Algorithms

Included algorithms follow the format of:

1. Input user, user.coordinates, user.trips, etc. objects from the database
2. Perform processing with a siblings `models.py` tightly coupled to the processing script. The methods should be primarily getters/setters

and helper methods, the core processing should try to occur entirely in the algorithm scripts to help keep the intention of each s

3. Algorithm models should be "wrapped for datakit" which is mapping the algorith's generally more complex model to the library's base

models for general usage. The base model should be as simple as possible to limit what must be stored in the database; if an attribute like duration can be inferred from a *start* and *end* timestamp, this is preferred over storing the value.

8.1.2 Pull Commits

Python black is used to format all library code into a consistent format. Note that two options are used:

- line limit of 120 characters (-l 120)
- skip string normalization (-S, explained below)

```
$ black -l 120 -S -t py37 tripkit
```

The string formatting exception to Python blacks default rules is to better convey meaning about intention within the library. Internal strings such as dictionary keys and filenames use single-quote (') endings whereas strings that will print to the user such as log messages use double-quote (") endings.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

tripkit.database, 18

Index

B

bulk_insert() (*tripkit.database.Database method*), 18

C

check_setup() (*tripkit.TripKit method*), 13
clear_trips() (*tripkit.database.Database method*), 18
count_users() (*tripkit.database.Database method*), 18
create() (*tripkit.database.Database method*), 18
csv (*tripkit.TripKit attribute*), 13
CSVIO (*class in tripkit.io*), 15

D

Database (*class in tripkit.database*), 18
database (*tripkit.TripKit attribute*), 13
DaySummary (*class in tripkit.models*), 22
delete_user_from_table() (*tripkit.database.Database method*), 19
drop() (*tripkit.database.Database method*), 19

G

generate_null_survey() (*tripkit.csvparser.ItinerumCSVParser method*), 14
generate_null_survey() (*tripkit.csvparser.QstarzCSVParser method*), 15
GeoJSONIO (*class in tripkit.io*), 17
GeopackageIO (*class in tripkit.io*), 17
get_uuid() (*tripkit.database.Database method*), 19

I

IO (*class in tripkit.io*), 15
io (*tripkit.TripKit attribute*), 13
ItinerumCSVParser (*class in tripkit.csvparser*), 14

L

load_activity_locations() (*tripkit.database.Database method*), 19
load_export_cancelled_prompt_responses() (*tripkit.csvparser.ItinerumCSVParser method*), 14
load_export_coordinates() (*tripkit.csvparser.ItinerumCSVParser method*), 15
load_export_coordinates() (*tripkit.csvparser.QstarzCSVParser method*), 15
load_export_prompt_responses() (*tripkit.csvparser.ItinerumCSVParser method*), 15
load_export_survey_responses() (*tripkit.csvparser.ItinerumCSVParser method*), 15
load_subway_entrances() (*tripkit.database.Database method*), 19
load_trip_day_summaries() (*tripkit.database.Database method*), 19
load_trips() (*tripkit.csvparser.ItinerumCSVParser method*), 15
load_trips() (*tripkit.database.Database method*), 19
load_user() (*tripkit.database.Database method*), 19
load_user_by_orig_id() (*tripkit.TripKit method*), 13
load_user_locations() (*tripkit.csvparser.QstarzCSVParser method*), 15
load_users() (*tripkit.TripKit method*), 14

P

process (*tripkit.TripKit attribute*), 14

Q

QstarzCSVParser (*class in tripkit.csvparser*), 15

S

save_trip_day_summaries() (*tripkit.database.Database method*), 19
save_trips() (*tripkit.database.Database method*), 19
setup() (*tripkit.TripKit method*), 14

T

Trip (*class in tripkit.models*), 21
TripKit (*class in tripkit*), 13
tripkit.database (*module*), 18
TripPoint (*class in tripkit.models*), 22

U

User (*class in tripkit.models*), 21

W

write_activities_daily() (*tripkit.io.CSVIO method*), 15
write_activity_locations() (*tripkit.io.GeoJSONIO method*), 17
write_activity_locations() (*tripkit.io.GeopackageIO method*), 17
write_activity_summaries() (*tripkit.io.CSVIO method*), 16
write_complete_days() (*tripkit.io.CSVIO method*), 16
write_condensed_activity_locations() (*tripkit.io.CSVIO method*), 16
write_condensed_trip_summaries() (*tripkit.io.CSVIO method*), 16
write_inputs() (*tripkit.io.GeoJSONIO method*), 17
write_inputs() (*tripkit.io.GeopackageIO method*), 18
write_mapmatch() (*tripkit.io.GeoJSONIO method*), 17
write_trip_summaries() (*tripkit.io.CSVIO method*), 16
write_trips() (*tripkit.io.CSVIO method*), 17
write_trips() (*tripkit.io.GeoJSONIO method*), 17
write_trips() (*tripkit.io.GeopackageIO method*), 18