Cellular Tracking Technologies: Data Tools

Jessica Gorzo

2021-04-29

Contents

Prerequisites

A RStudio tutorial is beyond the scope of this readme, but there are great resources to get you started with installing R and RStudio.

How to use GitHub

1. Create an account.
2. Work through chapters 6-12 here if you need to install git, and connect it all with RStudio:
3. Follow the instructions (at least through 5) here under "How to do this using RStudio and GitHub?"

- you don't need to enter the backticks in the shell
- this example is a bit misleading because it doesn't include the .git, copy the link to the clipboard like before
- RESTART RSTUDIO BEFORE MOVING ONTO STEP 6 IN THIS TUTORIAL

4. If you want to pull updates from here to your copy, see chapter 31.

Chapter 1

API

1.1 Install Postgres

If you choose to create a database out of your data (fair warning: in the future, the analysis tools will be based on this structure) you will need to install PostgreSQL on your machine.
1. For simplicity, set your Postgres user name to be the same as your computer user name. Otherwise, you will need to pass it as an argument to the connection
2. Create a database in Postgres owned by that user name. You may have to set a password, and you may have to pass that password as an argument to the connection

1.2 R script

1. As with the other R tools, I would suggest creating your own copy of "api_run.R" within your local repository, and modify that file.
2. Set your "outpath" variable to wherever your files will live. If you have already been manually downloading files, use that as your "outpath."

- The script will search that directory, and will only download files you haven't already downloaded.
- It will create a nested folder structure within that directory in the following order: project name, station(s), file types, files

3. If you do not want to create a database…

- comment out lines 8, 13-14
- remove the "conn" argument from the get_my_data() function (line 11)

4. If you do want to create a database locally, set "db_name" to the name of the Postgres database you created (line 7)

1.3 Terminal

Run “Rscript <path to your copy of api_run.R>” to run the script outside of RStudio (recommended)

Chapter 2

Start Here: Example Scripts

- "example.R" shows you example implementations of the data management and node health functions (also read comments, functions that produce files are commented out)
- "locate_example.R" is a template script for running the location functions

I suggest making your own copy of these scripts, renaming them, and modifying them with your file path inputs.

Chapter 3

About the Functions

There is a sub-folder within this repo named "functions" which is full of, well, scripts that contain functions! You'll notice they're often called (via source()) at the top of the example scripts. This loads in the custom functions that I have written to handle CTT data. Ultimately, these will be rolled into an R package.

3.1 Data Manager

3.1.1 load_data(infile)

Input
The input folder ("infile") can contain any melange of raw downloaded files from the sensor station (beep data, node health, GPS) all in the same folder or subfolders. Zipped folders need to be unzipped, but compressed files do not (i.e. csv.gz files are just fine as they are).

Output
The function will return a list of 3 dataframes from the files in the folder you give it:
1. beep data
2. node health
3. GPS

3.2 Node Health

3.2.1 node_channel_plots(health, freq)

This function is the "engine" behind the export function. You can run it standalone with the following parameters, but you don't have to.

Input
1. health: the 2nd dataframe output by the load_data() function
2. freq: the time interval for which you want variables to be summarized

Output The output is a nested list for each combination of channel and node, with the following plots for each:

1. battery
2. RSSI
3. number of check-ins
4. scaled number of check-ins as line plot over scaled RSSI
5. box plot of node RSSI

3.2.2 v2_plots(health, freq)

Input
1. health: the 2nd dataframe output by the load_data() function
2. freq: the time interval for which you want variables to be summarized

Output The output is a nested list for each combination of channel and node, with the following plots for each:

1. latitude
2. longitude
3. RSSI
4. dispersion

3.2.3 node_plots(health, nodes, freq)

NOTE: THIS ONLY WORKS FOR V2
Input
1. health: the 2nd dataframe output by the load_data() function
2. nodes: list of nodes
3. freq: the time interval for which you want variables to be summarized

Output The output is a nested list for each node, with the following plots for each:

1. RSSI
2. number of check-ins

3. battery

4. time mismatches

5. small time mismatches

3.2.4 gps_plots(gps, freq)

Input
1. gps: the 3rd data frame from the load_data() function
2. freq: the time interval of summary

Output
1. altitude
2. number of fixes

3.2.5 export_node_channel_plots(health_data, freq, out_path, x, y, z)

Input
1. health_data: the 2nd dataframe output by the load_data() function
2. freq: the time interval for which you want variables to be summarized
3. out_path: where you want your plots to go
4. x: the plot for the 1st panel
5. y: the plot for the 2nd panel
6. z: the plot for the 3rd panel

To assign x, y and z, look at the description for node_channel_plots() and select those plot indices in the order you want them on the page.

3.2.6 export_node_plots(health_data, freq, out_path, x, y, z)

NOTE: THIS ONLY WORKS FOR V2
same as above; indices for the plots can be chosen from the list under the node_plots() description