Introduction to places

2021 April 03

Places allows you to process GPS coordinates to extract meaningful stops and semantic labels (e.g., home, restaurant, gym).

Example Dataset

To explore the basic processing capabilities of places, we'll use the dataset places_gps.

This dataset represents data that may be collected from a project collecting smartphone sensing and ecological momentary assessments (EMA, i.e., longitudinal survey reports).

This dataset uses hypothetical data generated by the package author.

```
> head(places_gps)
```

	user_id	time_local	lat	lng	ema	Response.Time	tz_olson_id
1	123	1/24/21 17:01	28.74848	-80.04256	1	1/24/21 17:04	EST
2	123	1/24/21 20:25	28.49993	-80.97283	2	1/24/21 21:18	EST
3	123	1/24/21 20:45	28.45361	-81.31214	2	1/24/21 21:18	EST
4	123	1/24/21 20:54	28.41166	-81.56271	2	1/24/21 21:18	EST
5	123	1/24/21 21:04	28.41931	-81.58091	2	1/24/21 21:18	EST
6	123	1/24/21 23:02	28.41930	-81.58117	3	1/24/21 23:59	EST

The above dataframe includes the following variables:

- **user_id** = unique identifier for each participant
- **time_local** = datetime of GPS coordinates using the local time zone (not UTC)
- **lat** = latitude
- **lng** = longitude
- **ema** = survey report id
- **Response.Time** = datetime that survey report was submitted by participant
- **tz_olson_id** = time zone label

Depending on the functions you choose to run, not all of the above variables will be needed.

> **Important!** Your dataset may have datetime information stored as UTC (universal time) and/or local time (e.g., EST). You may be able to use either type of datetime variable, but you should **always** check the function's results to ensure that the output is what you expect. Remember, datetime variables may not behave as you expect.

Before getting started, let's update the datetime variables to POSIXct.

```
> places_gps$time_local <- as.POSIXct(strptime(places_gps$time_local,
        "%m/%d/%y %H:%M"), tz="UTC")
> places_gps$Response.Time <- as.POSIXct(strptime(places_gps$Response.Time,
        "%m/%d/%y %H:%M"), tz="UTC")
```

Important! I like to set the timezone, but you should decide what works best for your data.

Available Functions

- **get_clusters()** will cluster a dataframe of GPS coordinates into meaningful stops
- **get_home()** will identify which stop is likely to be the person's home
- **get_places()** uses the GoogleMaps API to label meaningful stops with semantic categories (e.g., café, gym, library)

Depending on the state of your dataset, you may not need to run all of the functions. If you do, you should run the functions in the above order.

Get_Clusters()

Before using the functions, the dataset must be prepared. Let's see what happens when we run the first function without preparing the dataset.

```
> clusters <- get_clusters(places_gps)
```

```
[1] "Please check your colnames are named according to the Dataframe
Requirements noted in documentation. Use ?get_clusters() for more information."
```

We get an error that we do not have correctly named columns. We check the documentation and update the names as required.

```
> colnames(places_gps)[c(2,4)] <- c("start_time", "lon")
```

Please note that it doesn't matter for this function whether start_time represents UTC or local time. If you have both datetime variables, we recommend labelling the UTC variable as "start_time" and the local time variable as "time_local".

Let's rerun the function! But first, please note that we have the option of updating the following:

- **max.accu** = an integer in meters. This number means there's a 68% probability that the true location is within this radius. The default is 165 m. Any GPS rows

with an accuracy higher than this will be dropped. If your dataset does not have accuracy data, this can be ignored.

- **max.speed** = an integer in meters/sec. It is the threshold value that distinguishes a row as Static or Moving. The default is 2.6 meters/sec. The default was chosen according to prior research findings that this is the upper gait speed for young adults. If your dataset has a column labelled "speed", the function will use this to determine the record's speed. If your dataset does not have a column labelled "speed", speed will be calculated as distance / time between two coordinates.
- **min.time** = an integer in minutes. It is the minimum amount of time between two points for the pair to be considered a stationary cluster. The defaults is 3 minutes. If you have a sequence of coordinates that identify the same location {GPS_1, GPS_2, GPS_3… GPS_N}, the first and last set of coordinates (GPS_1 and GPS_N) are used to determine if this threshold is met. If the time difference between GPS_1 and GPS_N is less than 3 minutes, it is assumed that the stop is not meaningful and it is not recorded as a stop. If the time difference between GPS_1 and GPS_N is greater than 3 minutes, it is assumed that the stop is meaningful and it is recorded as a stop.
- **max.time =** an integer in minutes. It is the maximum amount of time between two **consecutive** points for the pair to be considered a stationary cluster. The defaults is 15 minutes. Many smartphone sensing apps collect GPS data every 5 minutes. If the gap between two records is missing, it cannot be assumed that an individual was stationary in one stop.
- **max.distance =** an integer in meters. It is the maximum distance in meters between two points for the pair to be labelled a cluster. The defaults is 150 m.
- **var.segment =** if this variable is NOT set, clusters will be created based on the participant's entire dataset. If this variable is set, clusters will be segmented on the variable. A list can be provided. If the user sets var.segment, a character vector named **VS** is saved in the environment.

Please note, based on experience with datasets, I find that setting max.time is too conservative, so I will set max.time to something ridiculously large.

I will also set var.segment so that the function's output will return clusters matched with the participant's survey response. I will use the "ema" column to segment on. Alternatively, you could create additional variables to segment on hour (e.g., 1-2pm, 2-3pm, 3-4pm) or some other unit of time.

Please note, in this dataset, I could also segment on "Response.Time" or provide a list (e.g. c("ema", "Response.Time")), and I would get the same results. This is because "ema" and "Response.Time" are synonymous with each other and provide the same groupings.

The reason you might provide a list in this case is if you wanted to keep both variables in the output (see below – only "ema" is retained in the output since I only segmented on "ema").

Important! If you want to segment on multiple variables and they are not synonymous with each other, you should create a new variable that is a combination of the two. This is because certain functions will only use the first variable provided.

```
> new.max.time <- 60*60*24*365
> clusters <- get_clusters(places_gps, max.time = new.max.time, var.segment = "ema")
```

The function will return a list of two outputs. The first output is a dataframe that contains the moving records and stationary stops for all participants. Let's see what the results looks like.

```
> View(clusters[[1]])
```

user_id	ema	clust.final	lat.centroid.final	lon.centroid.final
123	2	999999	28.49993	-80.97283
123	2	999999	28.45361	-81.31214
123	2	999999	28.41166	-81.56271
123	2	999999	28.41931	-81.58091
123	3	1	28.41923	-81.58107
123	4	1	28.41923	-81.58107
123	5	1	28.41923	-81.58107

The first four rows contain records related to EMA survey 2 which could not be clustered. *Clust.final*, which represents the stop's ID, was therefore labelled with 999999, which means the record was not associated with a cluster. The next three rows contain records related to EMA surveys 3-5. The GPS records related to these EMAs could be clustered into meaningful stops. In this case, the participant responded to these 3 EMA surveys at the same stop, as indicated by clust.final (e.g., 1).

The lat.centroid.final and lon.centroid.final columns represent the weighted average of the latitude and longitude coordinates associated with a cluster. For clust.finals that are labelled 999999, the lat.centroid.final and lon.centroid.final records are simply the original latitude and longitude coordinates associated with that record. Please see how the lat.centroid.final and lon.centroid.final records in the first four rows are not the same, even though they are associated with the same EMA survey. On the other hand, the lat.centroid.final and lon.centroid.final records in the last three rows are the same—even though they are not associated with the same EMA survey—because they are associated with the same stop (e.g., 1).

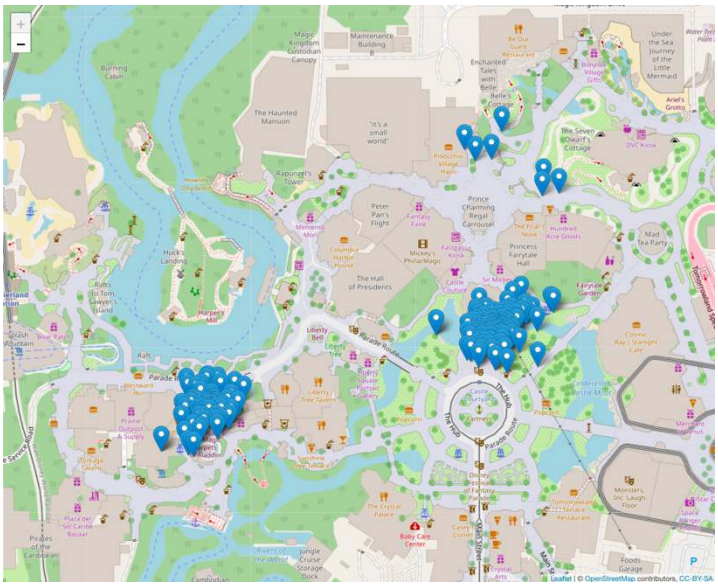
The second output is a list of dataframes for each participant (in this case, there is only 1 participant). There are a lot of variables in this dataframe – we will only look at a couple so you can see the difference between the two outputs.

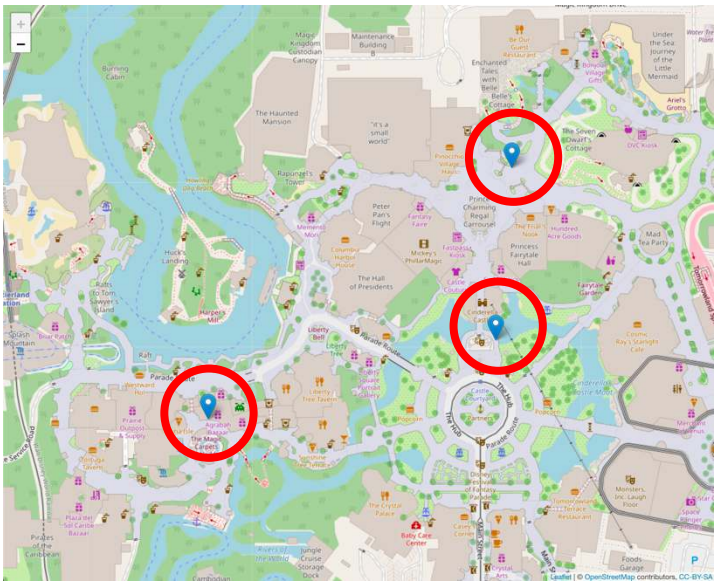
As you can see, the second output labels each of the original GPS records with the final cluster and weighted GPS coordinates. This is evident in the rows associated with EMA surveys 3-5.

> View(clusters1[[2]][[1]][, c(1:2,5,21,24,25)])

user_id	start_time	ema	clust.final	lat.centroid.final	lon.centroid.final
123	2021-01-24 20:25:00	2	999999	28.49993	-80.97283
123	2021-01-24 20:42:00	2	999999	28.45361	-81.31214
123	2021-01-24 20:54:00	2	999999	28.41166	-81.56271
123	2021-01-24 21:04:00	2	999999	28.41931	-81.58091
123	2021-01-24 23:02:00	3	1	28.41923	-81.58107
123	2021-01-24 23:22:00	3	1	28.41923	-81.58107
123	2021-01-24 23:42:00	3	1	28.41923	-81.58107
123	2021-01-24 23:57:00	3	1	28.41923	-81.58107
123	2021-01-25 01:38:00	4	1	28.41923	-81.58107
123	2021-01-25 02:08:00	4	1	28.41923	-81.58107
123	2021-01-25 04:08:00	5	1	28.41923	-81.58107
123	2021-01-25 04:18:00	5	1	28.41923	-81.58107
123	2021-01-25 04:50:00	5	1	28.41923	-81.58107

In order to better understand what we've done, please see the maps below. The first map contains all of the original latitude and longitude coordinates for the records identified as stationary. The second map contains the weighted latitude and longitude coordinates for these clusters.





Get_Home()

The get_home() function labels the stop which most frequently occurs during the night as the participant's home location. The default nighttime period is set as midnight to 6pm. However, tests on actual datasets showed that some participants may turn their phone off overnight, so you may want to consider expanding the nighttime period. For example, on this dataset, I ran the following:

```
> home <- get_home(places_gps, clusters[[1]], home.start = "21:30:00", home.end = "09:30:00")
```

> **Please note** that the function assumes the start_time variable is UTC. If this is not the case, your output will not be correct.

The function will return a list of two outputs. The first output is a dataframe that contains each participant's stops with the frequency of recorded visits during the nighttime hours. Let's see what the results looks like.

```
> View(home[[1]])
```

clust.final	lat.centroid	lon.centroid	clust.count	user_id
1	28.41923	-81.58108	28	123
2	28.41867	-81.58346	24	123

The second output is a dataframe that labels each participant's ema record as home, other, or in transit (though "in transit" may really be considered as in transit or unknown).

```
> View(home[[2]])
```

user_id	ema	clust.final	lat.centroid.final	lon.centroid.final	label
123	2	999999	NA	NA	In Transit
123	3	1	28.41923	-81.5811	Home
123	4	1	28.41923	-81.5811	Home
123	7	1	28.41923	-81.5811	Home
123	8	1	28.41923	-81.5811	Home
123	9	1	28.41923	-81.5811	Home
123	10	1	28.41923	-81.5811	Home
123	12	1	28.41923	-81.5811	Home
123	14	999999	NA	NA	In Transit
123	16	1	28.41923	-81.5811	Home
123	17	1	28.41923	-81.5811	Home
123	19	2	28.41865	-81.5835	Other
123	22	999999	NA	NA	In Transit
123	24	1	28.41923	-81.5811	Home
123	25	2	28.41865	-81.5835	Other

Get_Places()

To get semantic place labels for other locations, this package relies on the Googleway package which uses the GoogleMaps API. In order to use this function, you will need to create a Google account and obtain an API key from Google. Please be aware that this service may cost money.

The Googleway package will return information related to a location, such as place name and place type. The get_places() function is programmed to return information related to the closest matched location within a 50-meter radius of the stop's weighted latitude and longitude. You can adjust the radius by setting the variable radius within the get_places() function call. If you wish to retrieve all the placeNames and placeTypes associated with the stops in your dataset (and not just the single best guess retrieved with the get_places() function), you will need to use the Googleway package.

```
> key <- [YOUR_API_KEY_FROM_GOOGLE]
> labelled <- get_places(home[[2]], key)
```

Let's see the output!

user…	ema	clust…	lat…	lon..,	label	placeName	placeType	placeCat…
123	2	999999	NA	NA	In Transit	In Transit	In Transit	In Transit
123	3	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/ Bar
123	4	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/ Bar
123	7	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/ Bar
123	8	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/ Bar

123	9	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/Bar
123	10	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/Bar
123	12	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/Bar
123	14	999999	NA	NA	In Transit	In Transit	In Transit	In Transit
123	16	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/Bar
123	17	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/Bar
123	19	2	28.418	-81.583	Other	Prairie Outpost and Supply	food	Restaurant/Bar
123	22	999999	NA	NA	In Transit	In Transit	In Transit	In Transit
123	24	1	28.419	-81.581	Home	Cinderella's Royal Table	restaurant	Restaurant/Bar
123	25	2	28.418	-81.583	Other	Prairie Outpost and Supply	food	Restaurant/Bar
123	26	2	28.418	-81.583	Other	Prairie Outpost and Supply	food	Restaurant/Bar
123	28	2	28.418	-81.583	Other	Prairie Outpost and Supply	food	Restaurant/Bar

The variables to note include:

- **label** = the labels supplied after running get_home()
- **placeName** = the place name returned by Google API if a stop was identified. If a stop was not identified (e.g., clust.final = 999999), the placeName is returned as “In Transit” (though this may really be considered as in transit or unknown).
- **placeType** = the place type returned by Google API if a stop was identified. If a stop was not identified (e.g., clust.final = 999999), the placeType is returned as “In Transit” (though this may really be considered as in transit or unknown).
- **placeCategory** = using a dictionary defined by the package author, placeTypes are grouped into larger placeCategories.