Building formulas

Tom Stewart

2018-01-17

Helper functions

There are several helper functions included in **palmsplusr** that make creating formulas for fields easier.

Important ones are discussed below, but please also see the Function Reference or use the R help system `help(package=palmsplusr)` for more information.

palms_remove_tables()

`palms_remove_tables()`

This function removes all of the field tables from the global environment. It is useful to call this function before building any field tables so you don't run into duplicate errors. If you try to add a field that already exists, R will throw an error.

```
palms_add_field("weekday",    "dow < 6")
palms_add_field("weekday",    "dow < 6")
#> Error in palms_add_field("weekday", "dow < 6") : weekday already exists in palmsplus_fields

palms_remove_tables()
palms_add_field("weekday",    "dow < 6") #No error
```

The five **palmsplusr** field tables are:

1. `palmsplus_fields`
2. `palmsplus_domains`
3. `trajectory_fields`
4. `trajectory_locations`
5. `multimodal_fields`

palms_load_defaults()

`palms_load_defaults(epoch_length)`

This function adds basic fields to three of the five field tables (`palmsplus_fields`, `trajectory_fields`, and `multimodal_fields`). This does not add any fields to `palmsplus_domains` or `trajectory_locations` as these generally require external datasets.

The palms epoch length (in seconds) must be passed to this function so the `trajectory_fields` are setup correctly. This is achieved with the `palms_epoch()` function:

```
data("palms")

palms_remove_tables()
palms_load_defaults(palms_epoch(palms))
```

The default fields are shown in the tables below:

```
palmsplus_fields
```

Table 1: palmsplus_fields

name	formula	domain_field
weekday	dow < 6	FALSE
weekend	dow > 5	FALSE
indoors	iov == 3	FALSE
outdoors	iov == 1	FALSE
in_vehicle	iov == 2	FALSE
inserted	fixtypecode == 6	FALSE
pedestrian	tripmot == 1	FALSE
bicycle	tripmot == 2	FALSE
vehicle	tripmot == 3	FALSE
nonwear	activityintensity < 0	TRUE
wear	activityintensity >= 0	TRUE
sedentary	activityintensity == 0	TRUE
light	activityintensity == 1	TRUE
moderate	activityintensity == 2	TRUE
vigorous	activityintensity == 3	TRUE
mvpa	moderate + vigorous	TRUE

```
trajectory_fields
```

Table 2: trajectory_fields

name	formula	after_conversion
mot	first(tripmot)	FALSE
date	first(as.Date(datetime))	FALSE
start	datetime[triptype==1]	FALSE
end	datetime[triptype==4]	FALSE
duration	as.numeric(difftime(end, start, units = “secs”) + 15)	FALSE
nonwear	sum(activityintensity < 0) * 15	FALSE
wear	sum(activityintensity >= 0) * 15	FALSE
sedentary	sum(activityintensity == 0) * 15	FALSE
light	sum(activityintensity == 1) * 15	FALSE
moderate	sum(activityintensity == 2) * 15	FALSE
vigorous	sum(activityintensity == 3) * 15	FALSE
mvpa	moderate + vigorous	FALSE
length	as.numeric(st_length(.))	TRUE
speed	(length / duration) * 3.6	TRUE

```
multimodal_fields
```

Table 3: multimodal_fields

name	func
duration	sum
nonwear	sum
wear	sum
sedentary	sum
light	sum
moderate	sum
vigorous	sum
mvpa	sum
length	sum
speed	mean

palms_in_time()

```
palms_in_time(data, identifier, timetable, basis, start_col, end_col)
```

This function checks whether a `palms` point is between a two timestamps. Currently, this is hard-coded to require a class_timetable and participant_basis file as seen in the complete example article. This may become more generic in the future.

```
palms_in_time(., i, class_timetable, participant_basis, sch_start, sch_end)
```

You will notice the . and i in this formula, which represent the `data` and `identifier` parameters. See the last section of this article for an explaination.

palms_in_polygon()

```
palms_in_polygon(data, polygons, collapse_var = NULL)
```

This function checks whether a `palms` point falls inside a polygon. The polygon should have the same coordinate reference system as the palms dataset (ESPG: 4326). `MULTIPOLYGON` geometry is also supported, where each row in the shapefile represents more than one polygon. An example would be a city greenspace layer where all polygons are a single feature.

Alternatively, the optional `collapse_var` parameter can be used to dissolve polygons "on the fly". An example of this can be seen in the complete example article where participants may have more than one home.

By using the `identifier` as the `collapse_var`, the home.buffer is dissolved by identifier "on the fly":

```
palms_add_field("at_home", "palms_in_polygon(., filter(home.buffer, identifier == i), identifier)")
```

Again, you will notice the . and i in this formula. See below for an explaination.

Linking participant identifiers in palmsplus_fields

If you have read the complete example article, you may have noticed that some formulas contain a period (.) and 'i'.

palmsplusr field tables are evaluated using *dplyr*, the pipe opeator (*%>%*), and *tidy evaluation*. The period (.) represents the data argument in the helper functions. It works like this:

```
filter(palms, identifier = "BC0627")
# Is equivalent to:
palms %>% filter(., identifier = "BC0627") #palms is 'piped' in and becomes the .
```

The i is used to link the participant identifier (in the PALMS data) with a corresponding identifier in an external dataset (such as a shapefile).

An example of this is:

```
palms_in_polygon(., filter(home_poly, id_in_shapefile == i), id_in_shapefile)
```

The `palms_build_palmsplus()` function loops through each identifier one after the other. The i in this formula represents the participant identifier of the current iterration. The main workhorse loop of `palms_build_palmsplus()` is:

```
for (i in unique(data$identifier)) {
  x[[i]] <- data %>%
    filter(identifier == i) %>%
    mutate(!!! field_args) %>%
    mutate_if(is.logical, as.integer)
}
```

Notice:

- It is a for loop where i is the iterrator index
- The participants data is 'piped' into the mutate function, where all the fields are calculated. The . represents the data being piped into the mutate function.