Summary

2017-03-21

```r
vignette("Report", package = "lifelogr")
```

```r
library(lifelogr)
```

Lifelogging is the "process of tracking personal data generated by our own behavioral activities." Examples of lifelogging include tracking one's exercise and sleep. While devices like iPhones and Fitbits make collecting data on oneself easy, analyzing and interpreting that data is more difficult.

The lifelogr package makes life logging analysis and interpretation easy by:

- Providing a framework for working with multiple sources of personal data.
- Allowing users to easily create three sets of meaningful visualizations.
- Creating an ecosystem for experimentation - a feedback loop between self-data, and the process generating it.

Working with Multiple Sources of Data: the "Person" class

This package is built around the Person class, an R6 class. An object of Person class encapsulates a complete picture of a user, as seen through her data. The goal of this class is to make working with multiple sources of data as easy as possible by joining and standardizing multiple datasets. Users can then access the data from and across sources for visualization and experimentation.

Upon instantiation, a user can provide information about himself, the filepath to a csv file with Apple data, Fitbit account information, and/or dataframes of data from other sources.

Loading Multi-Source Data into Person Objects

Fitbit users can provide their account username and password upon instantiation. We use functions from the fitbitScraper package to collect all of a user's Fitbit data between the start and end date of interest.

```r
EX <- Person$new(fitbit_user_email = "example@domain.com",
                 fitbit_user_pw = "password",
                 apple_data_file = "apple.csv",
                 addl_data = NA,
                 user_info = list("name" = "EX", "age" = 29, "gender" = "male"),
                 target_steps = 10000,
                 group_assignments = list(data.frame(NA), data.frame(NA)),
                 start_date = "2017-01-19", end_date = "2017-02-17")
```

iPhone users can provide a filepath to a csv with their Apple health data. Users can download their data using an iOS app called QS Access (quantified self), which allows for selection of variables and time scales (by day or by hour), to include in a csv file. For greater compatibility with the lifelogr package, users are highly recommended to select the "by hour" time interval.

Users can provide data from other sources and self-tracking apps as pre-cleaned dataframes in the `addl_data` or `addl_data2` arguments. To make maximal use of lifelogr's functions, the column names of the same variables in the dataframe should match those referenced in the Person class, and class documentation (when the variables in the dataframe are the same as those referred to in the documentation).

Example of Person Class

EX is an example of a Person class for a user who provided their Fitbit account information upon instantiation. Functions within the Person class gathered and manipulated the data into data from three main "sources":

- `fitbit_daily` has Fitbit data recorded on a daily basis
- `fitbit_intraday` has Fitbit data recorded on an intraday basis, generally every 15 minutes, although for certain variables, such as heart rate, data are recorded every 5 minutes.
- `fitbit_util` has information about the dates in the range specified by the user upon Person instantiation, such as which day of the week a specific date is.

Dates, Times, and Date-Times

Dates and times come in numerous forms. lifelogr recognizes three types of dates/times as acceptable time-related variables to join datasets on:

- date: "2017-03-20"
- time: "08:00:00"
- datetime: "2017-03-20 08:00:00", a combination of a date and a time

Upon instantiation, the Person class automatically converts and renames Fitbit and Apple health data to adhere to this variable naming and format convention. If users provide their own addl_data as dataframes, they will need to have one or more of these time-related variables as columns in their dataframes in order to be able to join across their data source, and Fitbit/Apple health data.

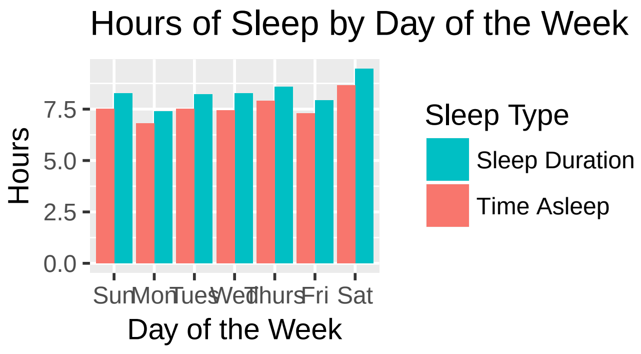
Visualization

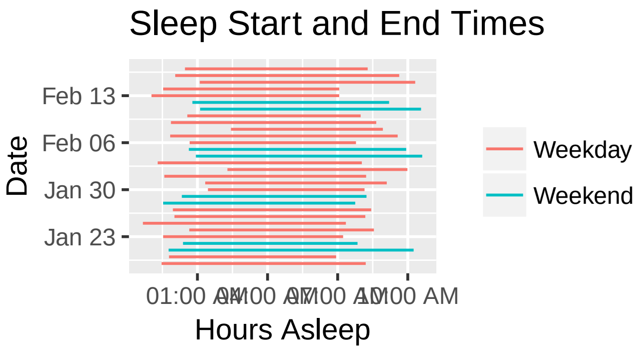
There are 3 functions which allow for a series of plots with just one function call: `plot_sleep_all`, `plot_daily_all`, and `plot_intraday_all`. Each acts like the `plot.lm` function, where users must click "enter" to see the next plot.

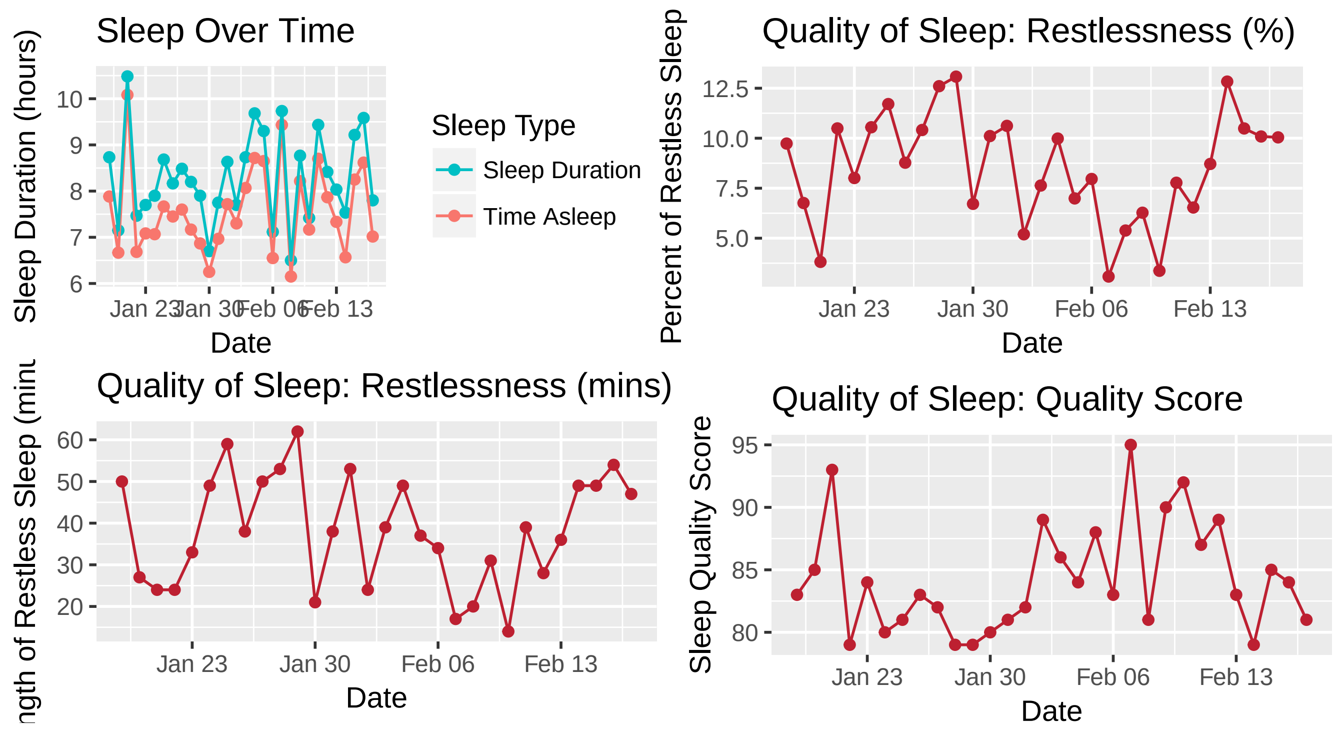
Each plot within each of the three series can also be generated individually with a call to its appropriate plot function.

For example, using `plot_sleep_all` to generate all the sleep plots for EX:

```
plot_sleep_all(EX)
#> Press [enter] to continue
#> Press [enter] to continue
#> Press [enter] to continue
#> Press [enter] to continue
#> Press [enter] to continue
#> Press [enter] to continue
```





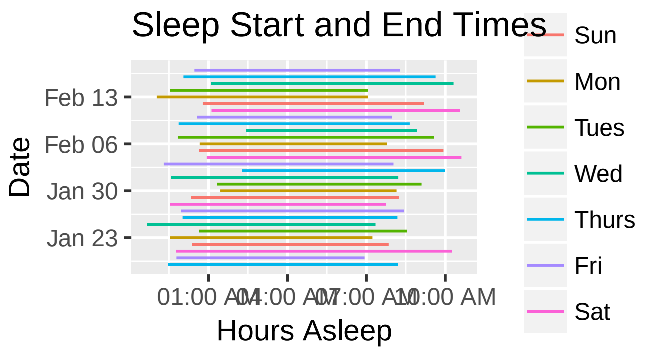


Calling `plot_daily_all` and `plot_intraday_all` will result in similar series of plots.

Users can also call each function individually using `plot_sleep(person, plot_type)` (or `plot_daily` or `plot_intraday`), and also pass in other arguments for plots with other arguments.

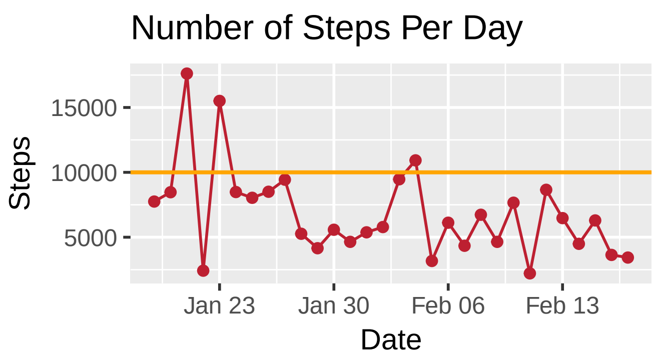
For example:

```r
plot_sleep(EX, "by_start_end_time", "day_of_week")
```



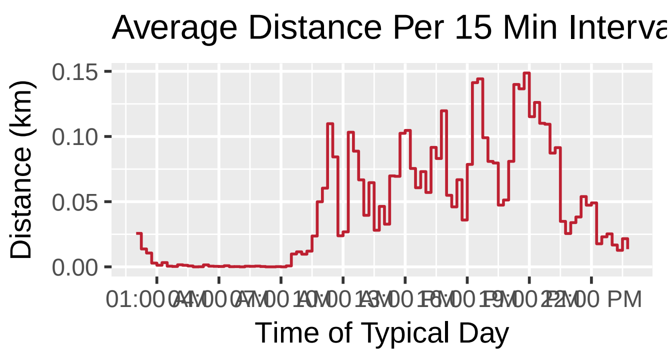
Using `plot_daily`:

```r
plot_daily(EX, "steps")
```



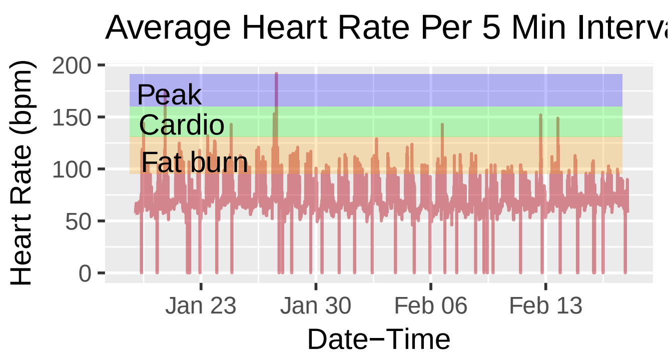
Using `plot_intraday`: The default is for `plot_intraday` to group the data by time intervals within each day so that data for a "typical day" is displayed.

```r
plot_intraday(EX, "distance", unit = "km")
```



However, it is also possible to specify that the plots use the raw data and plot over all datetimes.

```r
plot_intraday(EX, "bpm", FALSE)
```



Experimentation Framework

While almost all the visualizations shown above had date, date-time, or time on the x axis, the experimentation framework allows for greater flexibility. Users can select any variables of interest and examine their relationship.

Users can specify

- x variables of interest
- y measures of interest
- type of analysis ("plot", "correlation", "anova", "compare_groups", or "regression")
- time variable (date, time, or datetime)

The `experiment` function studies the effect of each variable, or all variables together (depending on the analysis), of the variable (X) and measure variables (Y) given.

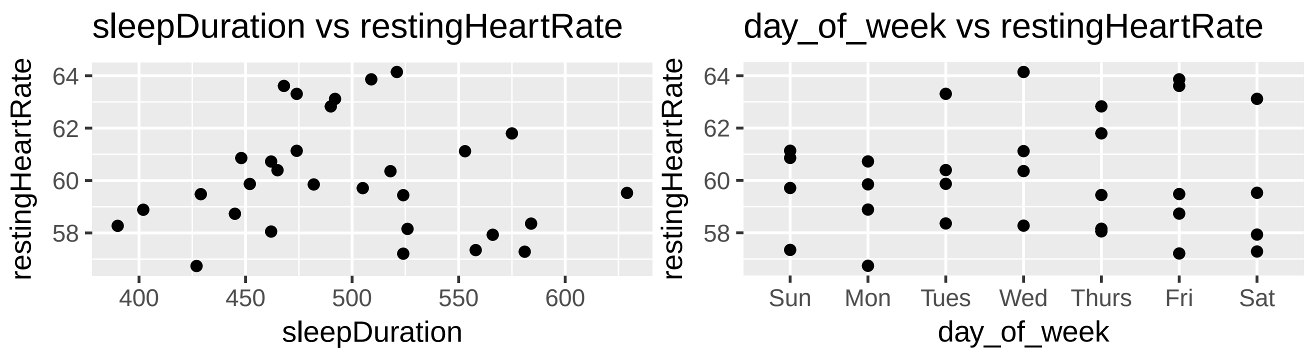
Users can also perform each analysis using the functions `l_plot`, `correlation`, `l_anova`, `compare_groups`, or `l_regression` instead of the higher-level function `experiment`.

The analysis functions below perform specific analyses, but using the standardized datasets in the fields of the Person class, the `create_dataset` function, which joins datasets across source types, and the templates provided by these analysis functions, users can easily perform their own unique analyses.

1. Plots

   To plot the relationship between sleep duration and resting heart rate and the relationship between day of week and resting heart rate on a date basis:

```
experiment(person = EX,
           variables = list("fitbit_daily" = c("sleepDuration"),
                            "util" = c("day_of_week")),
           measures = list("fitbit_daily" = c("restingHeartRate")),
           analysis = c("plot"),
           time_var = c("date"))
```



2. Correlation

   To calculate the correlation between sleep duration and number of steps on a date basis:

```
experiment(person = EX,
           variables = list("fitbit_daily" = c("sleepDuration")),
           measures = list("fitbit_intraday" = c("distance")),
           analysis = c("correlation"),
           time_var = c("date"))
#> [1] -0.01030084
```

   `l_plot` will generate the same result:

```
    dataset <- create_dataset(person = EX,
                              all_variables = list("fitbit_daily" = c("sleepDuration"),
                                                   "fitbit_intraday" = c("distance")),
                              time_var = c("date"))

    correlation_df <- correlation(dataset, person = EX,
                                  variables = list("fitbit_daily" = c("sleepDuration")),
                                  measures = list("fitbit_intraday" = c("distance")),
                                  time_var = "date")
#> [1] -0.01030084
```

3. ANOVA

To create ANOVAs for the effect of sleep duration and steps on resting heart rate on a date basis:

```
experiment(person = EX,
           variables = list("fitbit_daily" = c("sleepDuration", "steps")),
           measures = list("fitbit_daily" = c("restingHeartRate")),
           analysis = c("anova"),
           time_var = c("date"))
#> [1] "restingHeartRate ~ (sleepDuration + steps)^2"
#> Analysis of Variance Table
#>
#> Response: restingHeartRate
#>                     Df  Sum Sq Mean Sq F value Pr(>F)
#> sleepDuration        1   0.252  0.2525  0.0489 0.8266
#> steps                1   0.228  0.2277  0.0441 0.8352
#> sleepDuration:steps  1   0.203  0.2026  0.0393 0.8444
#> Residuals           26 134.102  5.1578
```

4. Compare Groups

   To compare across time variables not already defined, such as months, users can set up groups. Groups must be part of the Person instance, so it is oftentimes easier to just use `compare_groups`, which instead allows users to pass in new dataframes defining groupings.

   In this case, only data from January and February is in the sample instance of Person, so only two groups are compared:

```
 dataset <- create_dataset(person = EX,
                      all_variables = list("util" = c("month"),
                                           "fitbit_daily" =
                                             c("sleepDuration",
                                               "steps",
                                               "restingHeartRate")),
                      time_var = c("date"))

indiv_months <- data.frame("month" = c("Jan", "Feb", "Mar", "Apr", "May",
                                       "Jun", "Jul", "Aug",
                                       "Sep", "Oct", "Nov", "Dec"),
                           "group" = c(1:12))

compare_groups(dataset, person = EX,
                     addl_grouping_assignments = list("indiv_months" =
                                                        indiv_months),
                     names_of_groupings = c("indiv_months"),
                     variables_to_compare = c("steps", "restingHeartRate"))
#> [1] "month"
#> [1] "steps"
#> # A tibble: 2 × 3
#>   group     mean       sd
#>   <int>    <dbl>    <dbl>
#> 1     1 8142.692 4300.470
#> 2     2 5847.412 2350.607
#> [1] "restingHeartRate"
#> # A tibble: 2 × 3
#>   group     mean       sd
#>   <int>    <dbl>    <dbl>
#> 1     1 61.36231 1.856939
#> 2     2 59.08167 1.855553
```

5. Regression

   To run a regression of resting heart rate on steps ˆ 2

```
experiment(person = EX,
               variables = list("fitbit_daily" = c("steps")),
               measures = list("fitbit_daily" = c("restingHeartRate")),
               analysis = c("regression"),
               time_var = c("date"))
#> [1] "restingHeartRate ~ (steps)^2"
#>
#> Call:
#> lm(formula = restingHeartRate ~ (steps)^2, data = dataset)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -3.3178 -1.7636 -0.3039  1.1040  4.0493
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 5.993e+01  8.972e-01  66.796   <2e-16 ***
#> steps       2.008e-05  1.174e-04   0.171    0.865
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.193 on 28 degrees of freedom
#> Multiple R-squared:  0.001044,   Adjusted R-squared:  -0.03463
#> F-statistic: 0.02927 on 1 and 28 DF,  p-value: 0.8654
```

Shiny Application

We provide a sample Shiny application for visualization of the EX object's data.

Reflections

In our proposal, we explained that our package would accomplish 5 tasks:

1. Matching data from multiple sources
2. Visualization
3. Analysis
4. Recommendations
5. Experimentation

After discovering the disorganized structure of Fitbit data, we realized our package would be most useful (and feasible) if we focused on data cleaning, joining, and standardization, instead of recommendations for users. In order to be useful, recommendations must be tailor-made for an individual and her lifestyle; instead, our package gives users the ability to interact with their own data, and use it to inform and choose their lifestyle.

We chose to create an R6 class to create Person objects representing each user. We built upon another package called fitbitScraper, which scrapes from Fitbit's website. Functions from this package only pull portions of the data. We had to create functions to pull as much of the data as possible and join it together.

We created a standardized format for the data, and functions to manipulate Apple health data into this format as well.

We successfully created functions allowing the user to easily visualize data relating to certain portions of his life and run a series of experiments. We created a framework for users to add whatever additional data they want and then analyze it.

Future work can involve providing recommendations. Creating recommendation functionality is difficult because it generally involves a human. A package can tell its user that his/her resting heart rate is too high or that s/he is not meeting his/her step count, but providing recommendations for how to solve that issue is more difficult. Fitbit does not claim to be a replacement for a doctor, and neither does this package. Instead, we hope that with this dataset, we can begin the conversation between a user, and her data.