Package ‘evsim’

October 3, 2024

Title Electric Vehicle Charging Sessions Simulation

Version 1.6.0

Description Simulation of Electric Vehicles charging sessions using Gaussian models, together with time-series power demand calculations. The simulation methodology is published in Cañigueral et al. (2023, ISBN:0957-4174) <doi:10.1016/j.eswa.2023.120318>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Imports MASS, dplyr, lubridate, purrr, rlang, tidyr, jsonlite, dygraphs, ggplot2

Suggests shiny, shinydashboard, shinyWidgets, spelling, testthat (>= 3.0.0)

Depends R (>= 2.10)

URL `https://mcanigueral.github.io/evsim/`, `https://github.com/mcanigueral/evsim/`

BugReports `https://github.com/mcanigueral/evsim/issues`

Language en-US

Config/testthat/edition 3

NeedsCompilation no

Author Marc Cañigueral [aut, cre] (<`https://orcid.org/0000-0001-9724-5829`>)

Maintainer Marc Cañigueral <`marc.canigueral@udg.edu`>

Repository CRAN

Date/Publication 2024-10-03 10:00:02 UTC

Contents

`adapt_charging_features` *Adapt charging features*

Description

Calculate connection and charging times according to energy, power and time resolution

Usage

```
adapt_charging_features(
  sessions,
  time_resolution = 15,
  power_resolution = 0.01
)
```

Arguments

`sessions` tibble, sessions data set in standard format marked by `{evprof}` package (see this article)

`time_resolution` integer, time resolution (in minutes) of the sessions' datetime variables

`power_resolution` numeric, power resolution (in kW) of the sessions' power

Details

All sessions' `Power` must be higher than `0`, to avoid `NaN` values from dividing by zero. The `ConnectionStartDateTime` is first aligned to the desired time resolution, and the `ConnectionEndDateTime` is calculated according to the `ConnectionHours`. The `ChargingHours` is recalculated with the values of `Energy` and `Power`, limited by `ConnectionHours`. Finally, the charging times are also calculated.

Value

tibble

Examples

```
suppressMessages(library(dplyr))

sessions <- head(evsim::california_ev_sessions, 10)

sessions %>%
  select(ConnectionStartDateTime, ConnectionEndDateTime, Power)

adapt_charging_features(
  sessions,
  time_resolution = 60,
  power_resolution = 0.01
) %>%
  select(ConnectionStartDateTime, ConnectionEndDateTime, Power)

adapt_charging_features(
  sessions,
  time_resolution = 15,
  power_resolution = 1
) %>%
  select(ConnectionStartDateTime, ConnectionEndDateTime, Power)
```

`add_charging_infrastructure` *Assign a charging station to EV charging sessions*

Description

Variable `ChargingStation` and `Socket`will be assigned to the `sessions` tibble with a name pattern being: `names_prefix` + "CHS" + number

Usage

```
add_charging_infrastructure(
  sessions,
  resolution = 15,
  min_stations = 0,
  n_sockets = 2,
  names_prefix = NULL,
  duration_th = 0
)
```

Arguments

`sessions`	tibble, sessions data set in standard format marked by `{evprof}` package (see this article)
`resolution`	integer, time resolution in minutes
`min_stations`	integer, minimum number of charging stations to consider
`n_sockets`	integer, number of sockets per charging station
`names_prefix`	character, prefix of the charging station names (optional)
`duration_th`	integer between 0 and 100, minimum share of time (in percentage) of the "occupancy duration curve" (see function `plot_occupancy_duration_curve`). This is used to avoid sizing a charging infrastructure to host for example 100 vehicles when only 5% of time there are more than 80 vehicles connected. Then, setting `duration_th` = 5 will ensure that we don't over-size the charging infrastructure for the 100 vehicles. It is recommended to find this value through multiple iterations.

Value

tibble

Examples

```
# Assign a `ChargingStation` to every session according to the occupancy
sessions_infrastructure <- add_charging_infrastructure(
  sessions = head(evsim::california_ev_sessions, 50),
  resolution = 60
)
print(unique(sessions_infrastructure$ChargingStation))

# Now without considering the occupancy values that only represent
# a 10% of the time
sessions_infrastructure <- add_charging_infrastructure(
  sessions = head(evsim::california_ev_sessions, 50),
  resolution = 60, duration_th = 10
)
print(unique(sessions_infrastructure$ChargingStation))
```

expand_sessions *Expand sessions along time slots*

Description

Every session in sessions is divided in multiple time slots with the corresponding Power consumption, among other variables.

Usage

```
expand_sessions(sessions, resolution)
```

Arguments

sessions	tibble, sessions data set in standard format marked by evprof package (see this article)
resolution	integer, time resolution (in minutes) of the time slots

Details

The Power value is calculated for every time slot according to the original required energy. The columns PowerNominal, EnergyRequired and FlexibilityHours correspond to the values of the original session, and not to the expanded session in every time slot. The column ID shows the number of the time slot corresponding to the original session.

Value

tibble

Examples

```
library(dplyr)

sessions <- head(evsim::california_ev_sessions, 10)
expand_sessions(
  sessions,
  resolution = 60
)
```

get_charging_rates_distribution *Charging rates distribution*

Description

Get charging rates distribution in percentages from a charging sessions data set

Usage

```
get_charging_rates_distribution(sessions, unit = "year", power_interval = NULL)
```

Arguments

`sessions`	tibble, sessions data set in standard format marked by `{evprof}` package (see this article)t
`unit`	character. Valid base units are `second`, `minute`, `hour`, `day`, `week`, `month`, `bimonth`, `quarter`, `season`, `halfyear` and `year`. It corresponds to `unit` parameter in `lubridate::floor_date` function.
`power_interval`	numeric, interval of kW between power rates. It is used to round the `Power` values into this interval resolution. It can also be `NULL` to use all the original `Power` values.

Value

tibble

Examples

```
get_charging_rates_distribution(evsim::california_ev_sessions, unit = "year")
```

get_custom_ev_model *Create the custom EV model*

Description

Get the EV model object of class `evmodel`

Usage

```
get_custom_ev_model(
  names,
  months_lst = list(1:12, 1:12),
  wdays_lst = list(1:5, 6:7),
  parameters_lst,
  connection_log,
  energy_log,
  data_tz
)
```

Arguments

`names`	character vector with the given names of each time-cycle model
`months_lst`	list of integer vectors with the corresponding months of the year for each time-cycle model
`wdays_lst`	list of integer vectors with the corresponding days of the week for each time-cycle model (week start = 1)
`parameters_lst`	list of tibbles corresponding to the GMM parameters of every time-cycle model
`connection_log`	logical, true if connection models have logarithmic transformations
`energy_log`	logical, true if energy models have logarithmic transformations
`data_tz`	character, time zone of the original data (necessary to properly simulate new sessions)

Value

object of class `evmodel`

Examples

```
# For workdays time cycle
workdays_parameters <- dplyr::tibble(
  profile = c("Worktime", "Visit"),
  ratio = c(80, 20),
  start_mean = c(9, 11),
  start_sd = c(1, 4),
  duration_mean = c(8, 4),
  duration_sd = c(0.5, 2),
  energy_mean = c(15, 6),
  energy_sd = c(4, 3)
)

# For weekends time cycle
weekends_parameters <- dplyr::tibble(
  profile = "Visit",
  ratio = 100,
  start_mean = 12,
  start_sd = 4,
```

```
  duration_mean = 3,
  duration_sd = 2,
  energy_mean = 4,
  energy_sd = 4
)

parameters_lst <- list(workdays_parameters, weekends_parameters)

# Get the whole model
ev_model <- get_custom_ev_model(
  names = c("Workdays", "Weekends"),
  months_lst = list(1:12, 1:12),
  wdays_lst = list(1:5, 6:7),
  parameters_lst = parameters_lst,
  connection_log = FALSE,
  energy_log = FALSE,
  data_tz = "Europe/Amsterdam"
)
```

---

get_demand *Time-series EV demand*

Description

Obtain time-series of EV demand from sessions data set

Usage

```
get_demand(
  sessions,
  dttm_seq = NULL,
  by = "Profile",
  resolution = 15,
  mc.cores = 1
)
```

Arguments

sessions	tibble, sessions data set in standard format marked by `{evprof}` package (see this article)
dttm_seq	sequence of datetime values that will be the `datetime` variable of the returned time-series data frame.
by	character, being 'Profile' or 'Session'. When `by='Profile'` each column corresponds to an EV user profile.
resolution	integer, time resolution (in minutes) of the sessions datetime variables. If `dttm_seq` is defined this parameter is ignored.

mc.cores	integer, number of cores to use. Must be at least one, and parallelization requires at least two cores.

Details

Note that the time resolution of variables `ConnectionStartDateTime` and `ChargingStartDateTime` must coincide with `resolution` parameter. For example, if a charging session in `sessions` starts charging at 15:32 and `resolution = 15`, the load of this session won't be computed. To solve this, the function automatically aligns charging sessions' start time according to `resolution`, so following the previous example the session would start at 15:30.

Value

time-series tibble with first column of type `datetime`

Examples

```
suppressMessages(library(lubridate))
suppressMessages(library(dplyr))

# Get demand with the complete datetime sequence from the sessions
sessions <- head(evsim::california_ev_sessions, 100)
demand <- get_demand(
  sessions,
  by = "Session",
  resolution = 60
)
demand %>% plot_ts(ylab = "EV demand (kW)")

# Get demand with a custom datetime sequence and resolution of 15 minutes
sessions <- head(evsim::california_ev_sessions_profiles, 100)
dttm_seq <- seq.POSIXt(
  as_datetime(dmy(08102018)) %>% force_tz(tz(sessions$ConnectionStartDateTime)),
  as_datetime(dmy(11102018)) %>% force_tz(tz(sessions$ConnectionStartDateTime)),
  by = "15 mins"
)
demand <- get_demand(
  sessions,
  dttm_seq = dttm_seq,
  by = "Profile",
  resolution = 15
)
demand %>% plot_ts(ylab = "EV demand (kW)")
```

---

get_evmodel_parameters

Get `evmodel` *parameters in a list*

Description

Every time cycle is an element of the returned list, containing a list with the user profile as elements, each one containing the ratio and the corresponding tables with the statistic parameters of connection and energy GMM.

Usage

```
get_evmodel_parameters(evmodel)
```

Arguments

`evmodel`	object of class `evmodel`

Value

list

Examples

```
get_evmodel_parameters(evsim::california_ev_model)
```

`get_evmodel_summary` *Get `evmodel` parameters in a list of summary tables*

Description

Every time cycle is an element of the returned list, containing a table with a user profile in every row and the mean and standard deviation values of the GMM variables (connection duration, connection start time and energy). If the energy models were built by charging rate, the average `mean` and `sd` are provided without taking into account different charging rates (this information is lost in this summary).

Usage

```
get_evmodel_summary(evmodel)
```

Arguments

`evmodel`	object of class `evmodel`

Value

list

Examples

```
get_evmodel_summary(evsim::california_ev_model)
```

`get_occupancy` *Time-series EV occupancy*

Description

Obtain time-series of simultaneously connected EVs from sessions data set

Usage

```
get_occupancy(
  sessions,
  dttm_seq = NULL,
  by = "Profile",
  resolution = 15,
  mc.cores = 1
)
```

Arguments

`sessions`	tibble, sessions data set in standard format marked by `{evprof}` package (see this article)
`dttm_seq`	sequence of datetime values that will be the `datetime` variable of the returned time-series data frame.
`by`	character, being 'Profile' or 'Session'. When `by='Profile'` each column corresponds to an EV user profile.
`resolution`	integer, time resolution (in minutes) of the sessions datetime variables. If `dttm_seq` is defined this parameter is ignored.
`mc.cores`	integer, number of cores to use. Must be at least one, and parallelization requires at least two cores.

Details

Note that the time resolution of variable `ConnectionStartDateTime` must coincide with `resolution` parameter. For example, if a charging session in `sessions` starts charging at 15:32 and `resolution = 15`, the load of this session won't be computed. To solve this, the function automatically aligns charging sessions' start time according to `resolution`, so following the previous example the session would start at 15:30.

Value

time-series tibble with first column of type `datetime`

Examples

```
library(lubridate)
library(dplyr)

# Get occupancy with the complete datetime sequence from the sessions
sessions <- head(evsim::california_ev_sessions, 100)
connections <- get_occupancy(
  sessions,
  by = "ChargingStation",
  resolution = 60
)
connections %>%
  plot_ts(ylab = "Vehicles connected", legend_show = "onmouseover")

# Get occupancy with a custom datetime sequence and resolution of 15 minutes
sessions <- head(evsim::california_ev_sessions_profiles, 100)
dttm_seq <- seq.POSIXt(
  as_datetime(dmy(08102018)) %>% force_tz(tz(sessions$ConnectionStartDateTime)),
  as_datetime(dmy(11102018)) %>% force_tz(tz(sessions$ConnectionStartDateTime)),
  by = "15 mins"
)
connections <- get_occupancy(
  sessions,
  dttm_seq = dttm_seq,
  by = "Profile"
)
connections %>%
  plot_ts(ylab = "Vehicles connected", legend_show = "onmouseover")
```

---

get_user_profiles_distribution

User profiles distribution

Description

Get the user profiles distribution from the original data set used to build the model

Usage

```
get_user_profiles_distribution(evmodel)
```

Arguments

evmodel	object of class evmodel

Value

tibble

Examples

```
get_user_profiles_distribution(evsim::california_ev_model)
```

`plot_occupancy_duration_curve`
Plot the occupancy duration curve

Description

This term is based on the "load duration curve" and is useful to see the behavior of occupancy over the time in your charging installation. The steeper the curve, the shorter the duration that higher number of connections are sustained. Conversely, the flatter the curve, the longer the duration that higher number of connections are sustained. This information is crucial for various purposes, such as infrastructure planning, capacity sizing, and resource allocation.

Usage

```
plot_occupancy_duration_curve(
  sessions,
  dttm_seq = NULL,
  by = "Profile",
  resolution = 15,
  mc.cores = 1
)
```

Arguments

`sessions`	tibble, sessions data set in standard format marked by `{evprof}` package (see this article)
`dttm_seq`	sequence of datetime values that will be the `datetime` variable of the returned time-series data frame.
`by`	character, being 'Profile' or 'Session'. When `by='Profile'` each column corresponds to an EV user profile.
`resolution`	integer, time resolution (in minutes) of the sessions datetime variables. If `dttm_seq` is defined this parameter is ignored.
`mc.cores`	integer, number of cores to use. Must be at least one, and parallelization requires at least two cores.

Value

ggplot

Examples

```
library(dplyr)

sessions <- head(evsim::california_ev_sessions_profiles, 100)
plot_occupancy_duration_curve(
  sessions,
  by = "Profile",
  resolution = 15
)
```

---

plot_ts *Interactive plot for time-series tibbles*

Description

First column of the df tibble must be a datetime or date variable. The rest of columns must be numeric of the same units. This functions makes use of dygraphs package to generate an HTML Dygraphs plot.

Usage

```
plot_ts(
  df,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  legend_show = "auto",
  legend_width = 250,
  group = NULL,
  width = NULL,
  height = NULL,
  ...
)
```

Arguments

df	data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric
title	character, title of the plot (accepts HTML code)
xlab	character, X axis label (accepts HTML code)
ylab	character, Y axis label (accepts HTML code)
legend_show	character, when to display the legend. Specify "always" to always show the legend. Specify "onmouseover" to only display it when a user mouses over the chart. Specify "follow" to have the legend show as overlay to the chart which follows the mouse. The default behavior is "auto", which results in "always" when more than one series is plotted and "onmouseover" when only a single series is plotted.

`legend_width`	integer, width (in pixels) of the div which shows the legend.
`group`	character, dygraphs group to associate this plot with. The x-axis zoom level of dygraphs plots within a group is automatically synchronized.
`width`	Width in pixels (optional, defaults to automatic sizing)
`height`	Height in pixels (optional, defaults to automatic sizing)
`...`	extra arguments to pass to `dygraphs::dyOptions` function.

Value

dygraph

Examples

```
suppressMessages(library(lubridate))
suppressMessages(library(dplyr))

# Get demand with the complete datetime sequence from the sessions
sessions <- head(evsim::california_ev_sessions, 100)
demand <- get_demand(
  sessions,
  by = "Session",
  resolution = 60
)
demand %>% plot_ts()
```

`read_ev_model` *Read EV model*

Description

Read an EV model JSON file and convert it to object of class `evmodel`

Usage

```
read_ev_model(file)
```

Arguments

`file`	path to the JSON file

Value

object of class `evmodel`

Examples

```
ev_model <- california_ev_model # Model of example

save_ev_model(ev_model, file = file.path(tempdir(), "evmodel.json"))

read_ev_model(file = file.path(tempdir(), "evmodel.json"))
```

---

save_ev_model *Save the EV model*

Description

Save the EV model object of class `evmodel` to a JSON file

Usage

```
save_ev_model(evmodel, file)
```

Arguments

`evmodel`	object of class `evmodel` (see this link for more information)
`file`	character string with the path or name of the file

Value

nothing but saves the `evmodel` object in a JSON file

Examples

```
ev_model <- california_ev_model # Model of example

save_ev_model(ev_model, file = file.path(tempdir(), "evmodel.json"))
```

---

simulate_sessions *Simulation of EV sessions*

Description

Simulate EV charging sessions given the `evmodel` object and other contextual parameters.

Usage

```
simulate_sessions(
  evmodel,
  sessions_day,
  user_profiles,
  charging_powers,
  dates,
  resolution
)
```

Arguments

`evmodel`	object of class `evmodel` built with `{evprof}` (see this link for more information)
`sessions_day`	tibble with variables `time_cycle` (names corresponding to `evmodel$models$time_cycle`) and `n_sessions` (number of daily sessions per day for each time-cycle model)
`user_profiles`	tibble with variables `time_cycle`, `profile`, `ratio` and optionally `power`. It can also be `NULL` to use the `evmodel` original user profiles distribution. The powers must be in kW and the ratios between 0 and 1. The user profiles with a value of `power` will be simulated with this specific charging power. If `power` is `NA` then it is simulated according to the ratios of next parameter `charging_powers`.
`charging_powers`	tibble with variables `power` and `ratio`. The powers must be in kW and the ratios between 0 and 1. This is used to simulate the charging power of user profiles without a specific charging power in `user_profiles` parameter.
`dates`	date sequence that will set the time frame of the simulated sessions
`resolution`	integer, time resolution (in minutes) of the sessions datetime variables

Details

Some adaptations have been done to the output of the Gaussian models: the minimum simulated energy is considered to be 1 kWh, while the minimum connection duration is 30 minutes.

Value

tibble

Examples

```
library(dplyr)
library(lubridate)

# Get the example `evmodel`
ev_model <- evsim::california_ev_model

# Simulate EV charging sessions, considering that the Worktime sessions
# during Workdays have 11 kW, while all Visit sessions charge at 3.7kW or
# 11kW, with a distribution of 30% and 70% respectively.
```

```
simulate_sessions(
  ev_model,
  sessions_day = tibble(
    time_cycle = c("Workday", "Weekend"),
    n_sessions = c(15, 10)
  ),
  user_profiles = tibble(
    time_cycle = c("Workday", "Workday", "Weekend"),
    profile = c("Visit", "Worktime", "Visit"),
    ratio = c(0.5, 0.5, 1),
    power = c(NA, 11, NA)
  ),
  charging_powers = tibble(
    power = c(3.7, 11),
    ratio = c(0.3, 0.7)
  ),
  dates = seq.Date(today(), today()+days(4), length.out = 4),
  resolution = 15
)
```

Index