Package 'evprof'

March 14, 2024

Title Electric Vehicle Charging Sessions Profiling and Modelling

Version 1.1.2

Description Tools for modelling electric vehicle charging sessions into generic groups with similar connection patterns called ``user profiles'', using Gaussian Mixture Models clustering. The clustering and profiling methodology is described in Cañigueral and Meléndez (2021, ISBN:0142-0615) <doi:10.1016/j.ijepes.2021.107195>.

License GPL-3

URL https://github.com/mcanigueral/evprof/, https://mcanigueral.github.io/evprof/

BugReports https://github.com/mcanigueral/evprof/issues

Depends R (>= 3.5.0)

Imports cowplot, dbscan, dplyr, ggplot2, jsonlite, lubridate, MASS, mclust, plotly, purrr, rlang, tibble, tidyr

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0), utils

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2024-03-14 14:50:05 UTC

R topics documented:

`choose_k_GMM` *Visualize BIC indicator to choose the number of clusters*

Description

The Baysian Information Criterion (BIC) is the value of the maximized loglikelihood with a penalty on the number of parameters in the model, and allows comparison of models with differing parameterizations and/or differing numbers of clusters. In general the larger the value of the BIC, the stronger the evidence for the model and number of clusters (see, e.g. Fraley and Raftery 2002a).

Usage

```
choose_k_GMM(
  sessions,
  k,
  mclust_tol = 1e-08,
  mclust_itmax = 10000,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

sessions	tibble, sessions data set in evprof standard format.
k	sequence with the number of clusters, for example 1:10, for 1 to 10 clusters.
mclust_tol	tolerance parameter for clustering
mclust_itmax	maximum number of iterations
log	logical, whether to transform ConnectionStartDateTime and ConnectionHours variables to natural logarithmic scale (base = exp(1)).
start	integer, start hour in the x axis of the plot.

Value

BIC plot

Examples

```
choose_k_GMM(california_ev_sessions, k = 1:4, start = 3)
```

---

cluster_sessions *Cluster sessions with* mclust *package*

Description

Cluster sessions with mclust package

Usage

```
cluster_sessions(
  sessions,
  k,
  seed,
  mclust_tol = 1e-08,
```

```
  mclust_itmax = 10000,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`k`	number of clusters
`seed`	random seed
`mclust_tol`	tolerance parameter for clustering
`mclust_itmax`	maximum number of iterations
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).
`start`	integer, start hour in the x axis of the plot.

Value

list with two attributes: sessions and models

Examples

```
library(dplyr)

# Select working day sessions (`Timecycle == 1`) that
# disconnect the same day (`Disconnection == 1`)
sessions_day <- california_ev_sessions %>%
  divide_by_timecycle(
    months_cycles = list(1:12), # Not differentiation between months
    wdays_cycles = list(1:5, 6:7) # Differentiation between workdays/weekends
  ) %>%
  divide_by_disconnection(
    division_hour = 10, start = 3
  ) %>%
  filter(
    Disconnection == 1, Timecycle == 1
  ) %>%
  sample_frac(0.05)
plot_points(sessions_day, start = 3)

# Identify two clusters
sessions_clusters <- cluster_sessions(
  sessions_day, k=2, seed = 1234, log = TRUE
)

# The column `Cluster` has been added
names(sessions_clusters$sessions)
plot_points(sessions_clusters$sessions) +
  ggplot2::aes(color = Cluster)
```

`cut_sessions` *Cut outliers based on minimum and maximum limits of ConnectionHours and ConnectionStartDateTime variables*

Description

Cut outliers based on minimum and maximum limits of ConnectionHours and ConnectionStartDateTime variables

Usage

```
cut_sessions(
  sessions,
  connection_hours_min = NA,
  connection_hours_max = NA,
  connection_start_min = NA,
  connection_start_max = NA,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`connection_hours_min`	numeric, minimum of connection hours (duration). If NA the minimum value is considered.
`connection_hours_max`	numeric, maximum of connection hours (duration). If NA the maximum value is considered.
`connection_start_min`	numeric, minimum hour of connection start (hour as numeric). If NA the minimum value is considered.
`connection_start_max`	numeric, maximum hour of connection start (hour as numeric). If NA the maximum value is considered.
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).
`start`	integer, start hour in the x axis of the plot.

Value

session dataframe

Examples

```
library(dplyr)
# Localize the outlying sessions above a certain threshold
california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_points(start = 3)

# For example sessions that start before 5 AM or that are
# longer than 20 hours are considered outliers
sessions_clean <- california_ev_sessions %>%
  sample_frac(0.05) %>%
  cut_sessions(
    start = 3,
    connection_hours_max = 20,
    connection_start_min = 5
  )
plot_points(sessions_clean, start = 3)
```

`define_clusters` *Define each cluster with a user profile interpretation*

Description

Every cluster has a centroid (i.e. average start time and duration) that can be related to a daily human behaviour or connection pattern (e.g. Worktime, Dinner, etc.). In this function, a user profile name is assigned to every cluster.

Usage

```
define_clusters(
  models,
  interpretations = NULL,
  profile_names = NULL,
  log = FALSE
)
```

Arguments

`models`	tibble, parameters of the clusters' GMM models obtained with function `cluster_sessions()` (object `models` of the returned list)
`interpretations`	character vector with interpretation sentences of each cluster (arranged by cluster number)
`profile_names`	character vector with user profile assigned to each cluster (arranged by cluster number)
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).

Value

tibble object

Examples

```
library(dplyr)

# Select working day sessions (`Timecycle == 1`) that
# disconnect the same day (`Disconnection == 1`)
sessions_day <- california_ev_sessions %>%
  divide_by_timecycle(
    months_cycles = list(1:12), # Not differentiation between months
    wdays_cycles = list(1:5, 6:7) # Differentiation between workdays/weekends
  ) %>%
  divide_by_disconnection(
    division_hour = 10, start = 3
  ) %>%
  filter(
    Disconnection == 1, Timecycle == 1
  ) %>%
  sample_frac(0.05)
plot_points(sessions_day, start = 3)

# Identify two clusters
sessions_clusters <- cluster_sessions(
  sessions_day, k=2, seed = 1234, log = TRUE
)

# Plot the clusters found
plot_bivarGMM(
  sessions = sessions_clusters$sessions,
  models = sessions_clusters$models,
  log = TRUE, start = 3
)

# Define the clusters with user profile interpretations
define_clusters(
  models = sessions_clusters$models,
  interpretations = c(
    "Connections during working hours",
    "Connections during all day (high variability)"
  ),
  profile_names = c("Workers", "Visitors"),
  log = TRUE
)
```

`detect_outliers` *Detect outliers*

Description

Detect outliers

Usage

```
detect_outliers(
  sessions,
  MinPts = NULL,
  eps = NULL,
  noise_th = 2,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`MinPts`	MinPts parameter for DBSCAN clustering
`eps`	eps parameter for DBSCAN clustering
`noise_th`	noise threshold
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).
`start`	integer, start hour in the x axis of the plot.

Value

sessions tibble with extra boolean column `Outlier`

Examples

```
library(dplyr)
sessions_outliers <- california_ev_sessions %>%
  sample_frac(0.05) %>%
  detect_outliers(start = 3, noise_th = 5, eps = 2.5)
```

`divide_by_disconnection` *Divide sessions by disconnection day*

Description

Divide sessions by disconnection day

Usage

```
divide_by_disconnection(
  sessions,
  division_hour,
  start = getOption("evprof.start.hour")
)
```

Arguments

sessions	tibble, sessions data set in evprof standard format.
division_hour	Hour to divide the groups according to disconnection time
start	integer, start hour in the x axis of the plot.

Value

same sessions data set with extra column "Disconnection"

Examples

```
library(dplyr)
sessions_disconnection <- california_ev_sessions %>%
  sample_frac(0.05) %>%
  divide_by_disconnection(
    start = 2, division_hour = 5
  )

# The column `Disconnection` has been added
names(sessions_disconnection)

library(ggplot2)
sessions_disconnection %>%
  tidyr::drop_na() %>%
  plot_points() +
  facet_wrap(vars(Disconnection))
```

---

divide_by_timecycle *Divide sessions by time-cycle*

Description

Divide sessions by time-cycle

Usage

```
divide_by_timecycle(
  sessions,
  months_cycles = list(1:12),
  wdays_cycles = list(1:5, 6:7),
  start = getOption("evprof.start.hour")
)
```

Arguments

sessions	tibble, sessions data set in evprof standard format.
months_cycles	list containing Monthly cycles
wdays_cycles	list containing Weekdays cycles
start	integer, start hour in the x axis of the plot.

Value

same sessions data set with extra column "Timecycle"

Examples

```
library(dplyr)
sessions_timecycles <- california_ev_sessions %>%
  sample_frac(0.05) %>%
  divide_by_timecycle(
    months_cycles = list(1:12),
    wdays_cycles = list(1:5, 6:7)
  )

# The column `Timecycle` has been added
names(sessions_timecycles)

library(ggplot2)
plot_points(sessions_timecycles) +
  facet_wrap(vars(Timecycle))
```

---

drop_outliers *Drop outliers*

Description

Drop outliers

Usage

```
drop_outliers(sessions)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.

Value

sessions without outliers nor column `Outlier`

Examples

```
library(dplyr)
sessions_outliers <- california_ev_sessions %>%
  sample_frac(0.05) %>%
  detect_outliers(start = 3, noise_th = 5, eps = 2.5)

plot_outliers(sessions_outliers, start = 3)

sessions_clean <- drop_outliers(sessions_outliers)

plot_points(sessions_clean, start = 3)
```

`get_charging_rates_distribution` *Get charging rates distribution in percentages*

Description

Get charging rates distribution in percentages

Usage

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

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`unit`	character, lubridate `floor_date` unit parameter

Value

tibble

Examples

```
get_charging_rates_distribution(california_ev_sessions, unit="month")
get_charging_rates_distribution(california_ev_sessions, unit="month")
```

`get_connection_models` *Get a tibble of connection GMM for every user profile*

Description

Get a tibble of connection GMM for every user profile

Usage

```
get_connection_models(
  subsets_clustering = list(),
  clusters_definition = list()
)
```

Arguments

`subsets_clustering`	list with clustering results of each subset (direct output from function `cluser_sessions()`)
`clusters_definition`	list of tibbles with clusters definitions (direct output from function `define_clusters()`) of each sub-set

Value

tibble

Examples

```
library(dplyr)

# Select working day sessions (`Timecycle == 1`) that
# disconnect the same day (`Disconnection == 1`)
sessions_day <- california_ev_sessions %>%
  divide_by_timecycle(
    months_cycles = list(1:12), # Not differentiation between months
    wdays_cycles = list(1:5, 6:7) # Differentiation between workdays/weekends
  ) %>%
  divide_by_disconnection(
    division_hour = 10, start = 3
  ) %>%
  filter(
    Disconnection == 1, Timecycle == 1
  ) %>%
  sample_frac(0.05)
plot_points(sessions_day, start = 3)

# Identify two clusters
sessions_clusters <- cluster_sessions(
  sessions_day, k=2, seed = 1234, log = TRUE
```

```
)

# Plot the clusters found
plot_bivarGMM(
  sessions = sessions_clusters$sessions,
  models = sessions_clusters$models,
  log = TRUE, start = 3
)

# Define the clusters with user profile interpretations
clusters_definitions <- define_clusters(
  models = sessions_clusters$models,
  interpretations = c(
    "Connections during working hours",
    "Connections during all day (high variability)"
  ),
  profile_names = c("Workers", "Visitors"),
  log = TRUE
)

# Create a table with the connection GMM parameters
get_connection_models(
  subsets_clustering = list(sessions_clusters),
  clusters_definition = list(clusters_definitions)
)
```

`get_daily_avg_n_sessions` *Get the daily average number of sessions given a range of years, months and weekdays*

Description

Get the daily average number of sessions given a range of years, months and weekdays

Usage

```
get_daily_avg_n_sessions(sessions, years, months, wdays)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`years`	vector of integers, range of years to consider
`months`	vector of integers, range of months to consider
`wdays`	vector of integers, range of weekdays to consider

Value

tibble with the number of sessions of each date in the given time period

Examples

```
get_daily_avg_n_sessions(
  california_ev_sessions,
  year = 2018, months = c(5, 6), wdays = 1
)
```

`get_daily_n_sessions` *Get daily number of sessions given a range of years, months and weekdays*

Description

Get daily number of sessions given a range of years, months and weekdays

Usage

```
get_daily_n_sessions(sessions, years, months, wdays)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`years`	vector of integers, range of years to consider
`months`	vector of integers, range of months to consider
`wdays`	vector of integers, range of weekdays to consider

Value

tibble with the number of sessions of each date in the given time period

Examples

```
get_daily_n_sessions(
  california_ev_sessions,
  year = 2018, months = c(5, 6), wdays = 1
)
```

`get_dbscan_params` *Get the minPts and eps values for DBSCAN to label only a specific percentage as noise*

Description

Get the minPts and eps values for DBSCAN to label only a specific percentage as noise

Usage

```
get_dbscan_params(
  sessions,
  MinPts,
  eps0,
  noise_th = 2,
  eps_offset_pct = 0.9,
  eps_inc_pct = 0.02,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`MinPts`	DBSCAN MinPts parameter
`eps0`	DBSCAN eps parameter corresponding to the elbow of kNN dist plot
`noise_th`	noise threshold
`eps_offset_pct`	eps_offset_pct
`eps_inc_pct`	eps_inc_pct
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).
`start`	integer, start hour in the x axis of the plot.

Value

tibble with minPts and eps parameters, and the corresponding noise

get_energy_models *Get a tibble of energy GMM for every user profile*

Description

This function simulates random energy values, makes the density curve and overlaps the simulated density curve with the real density curve of the user profile's energy values. This is useful to appreciate how the modeled values fit the real ones and increase or decrease the number of Gaussian components.

Usage

```
get_energy_models(sessions_profiles, log = TRUE, by_power = FALSE)
```

Arguments

sessions_profiles	tibble, sessions data set in evprof standard format with user profile attribute `Profile`
log	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).
by_power	Logical, true to fit the energy models for every charging rate separately

Value

tibble

Examples

```
library(dplyr)

# Classify each session to the corresponding user profile
sessions_profiles <- california_ev_sessions_profiles %>%
  dplyr::sample_frac(0.05)

# Get a table with the energy GMM parameters
get_energy_models(sessions_profiles, log = TRUE)

# If there is a `Power` variable in the data set
# you can create an energy model per power rate and user profile
# First it is convenient to round the `Power` values for more generic models
sessions_profiles <- sessions_profiles %>%
  mutate(Power = round_to_interval(Power, 3.7)) %>%
  filter(Power < 11)
sessions_profiles$Power[sessions_profiles$Power == 0] <- 3.7
get_energy_models(sessions_profiles, log = TRUE, by_power = TRUE)
```

`get_ev_model` *Get the EV model object of class* `evmodel`

Description

Get the EV model object of class `evmodel`

Usage

```
get_ev_model(
  names,
  months_lst = list(1:12, 1:12),
  wdays_lst = list(1:5, 6:7),
  connection_GMM,
  energy_GMM,
  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 model (week start = 1)
`connection_GMM`	list of different connection bivariate GMM obtained from `get_connection_models`
`energy_GMM`	list of different energy univariate GMM obtained from `get_energy_models`
`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

```
# The package evprof provides example objects of connection and energy
# Gaussian Mixture Models obtained from California's open data set
# (see California article in package website) created with functions
# `get_connection models` and `get_energy models`.

# For workdays sessions
workdays_connection_models <- evprof::california_GMM$workdays$connection_models
workdays_energy_models <- evprof::california_GMM$workdays$energy_models

# For weekends sessions
weekends_connection_models <- evprof::california_GMM$weekends$connection_models
weekends_energy_models <- evprof::california_GMM$weekends$energy_models

# Get the whole model
ev_model <- get_ev_model(
  names = c("Workdays", "Weekends"),
  months_lst = list(1:12, 1:12),
  wdays_lst = list(1:5, 6:7),
  connection_GMM = list(workdays_connection_models, weekends_connection_models),
  energy_GMM = list(workdays_energy_models, weekends_energy_models),
  connection_log = TRUE,
  energy_log = TRUE,
  data_tz = "America/Los_Angeles"
)
```

---

plot_bivarGMM *Plot Bivariate Gaussian Mixture Models*

Description

Plot Bivariate Gaussian Mixture Models

Usage

```
plot_bivarGMM(
  sessions,
  models,
  profiles_names = seq(1, nrow(models)),
  points_size = 0.25,
  lines_size = 1,
  legend_nrow = 2,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

sessions	tibble, sessions data set in evprof standard format.
models	tibble, parameters of the clusters' GMM models obtained with function cluster_sessions (object models of the returned list)
profiles_names	names of profiles
points_size	size of scatter points in the plot
lines_size	size of lines in the plot
legend_nrow	number of rows in legend
log	logical, whether to transform ConnectionStartDateTime and ConnectionHours variables to natural logarithmic scale (base = exp(1)).
start	integer, start hour in the x axis of the plot.

Value

ggplot2 plot

Examples

```
library(dplyr)

# Select working day sessions (`Timecycle == 1`) that
# disconnect the same day (`Disconnection == 1`)
sessions_day <- california_ev_sessions %>%
  divide_by_timecycle(
    months_cycles = list(1:12), # Not differentiation between months
    wdays_cycles = list(1:5, 6:7) # Differentiation between workdays/weekends
  ) %>%
  divide_by_disconnection(
    division_hour = 10, start = 3
  ) %>%
  filter(
    Disconnection == 1, Timecycle == 1
  ) %>%
  sample_frac(0.05)
plot_points(sessions_day, start = 3)

# Identify two clusters
sessions_clusters <- cluster_sessions(
  sessions_day, k=2, seed = 1234, log = TRUE
)

# Plot the clusters found
plot_bivarGMM(
  sessions = sessions_clusters$sessions,
  models = sessions_clusters$models,
  log = TRUE, start = 3
)
```

`plot_density_2D` *Density plot in 2D, considering Start time and Connection duration as variables*

Description

Density plot in 2D, considering Start time and Connection duration as variables

Usage

```
plot_density_2D(
  sessions,
  bins = 15,
  by = c("wday", "month", "year"),
  start = getOption("evprof.start.hour"),
  log = FALSE
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`bins`	integer, parameter to pass to `ggplot2::stat_density_2d`
`by`	variable to facet the plot. Character being "wday", "month" or "year", considering the week to start at wday=1.
`start`	integer, start hour in the x axis of the plot.
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).

Value

ggplot2 plot

Examples

```
library(dplyr)

california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_density_2D(by = "wday", start = 3, bins = 15, log = FALSE)
```

`plot_density_3D` *Density plot in 3D, considering Start time and Connection duration as variables*

Description

Density plot in 3D, considering Start time and Connection duration as variables

Usage

```
plot_density_3D(
  sessions,
  start = getOption("evprof.start.hour"),
  eye = list(x = -1.5, y = -1.5, z = 1.5),
  log = FALSE
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`start`	integer, start hour in the x axis of the plot.
`eye`	list containing x, y and z points of view. Example: `list(x = -1.5, y = -1.5, z = 1.5)`
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).

Value

plotly plot (html)

Examples

```
library(dplyr)
california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_density_3D(start = 3)
```

plot_division_lines *Iteration over evprof::plot_division_line function to plot multiple lines*

Description

Iteration over evprof::plot_division_line function to plot multiple lines

Usage

```
plot_division_lines(ggplot_points, n_lines, division_hour)
```

Arguments

ggplot_points	ggplot2 returned by evprof::plot_points function
n_lines	number of lines to plot
division_hour	Hour to divide the groups according to disconnection time

Value

ggplot2 function

Examples

```
library(dplyr)
california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_points(start = 3) %>%
  plot_division_lines(n_lines = 1, division_hour = 5)
```

plot_energy_models *Compare density of estimated energy with density of real energy vector*

Description

Compare density of estimated energy with density of real energy vector

Usage

```
plot_energy_models(energy_models, nrow = 2)
```

Arguments

energy_models	energy models returned by function get_energy_models
nrow	integer, number of rows in the plot grid (passed to cowplot::plot_grid)

Value

ggplot

Examples

```
# The package evprof provides example objects of connection and energy
# Gaussian Mixture Models obtained from California's open data set
# (see California article in package website) created with functions
# `get_connection models` and `get_energy models`.

# Get the working days energy models
energy_models <- evprof::california_GMM$workdays$energy_models

# Plot energy models
plot_energy_models(energy_models)
```

`plot_histogram` *Histogram of a variable from sessions data set*

Description

Histogram of a variable from sessions data set

Usage

```
plot_histogram(sessions, var, binwidth = 1)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`var`	character, column name to compute the histogram for
`binwidth`	integer, with of histogram bins

Value

ggplot plot

Examples

```
plot_histogram(california_ev_sessions, "Power", binwidth = 2)
plot_histogram(california_ev_sessions, "Power", binwidth = 0.1)
```

plot_histogram_grid *Grid of multiple variable histograms*

Description

Grid of multiple variable histograms

Usage

```
plot_histogram_grid(
  sessions,
  vars = evprof::sessions_summary_feature_names,
  binwidths = rep(1, length(vars)),
  nrow = NULL,
  ncol = NULL
)
```

Arguments

sessions	tibble, sessions data set in evprof standard format.
vars	vector of characters, variables to plot
binwidths	vector of integers, binwidths of each variable histogram. The length of the vector must correspond to the length of the vars parameter.
nrow	integer, number of rows of the plot grid
ncol	integer, number of columns of the plot grid

Value

grid plot

Examples

```
plot_histogram_grid(california_ev_sessions)
plot_histogram_grid(california_ev_sessions, vars = c("Energy", "Power"))
```

plot_kNNdist *Plot kNNdist*

Description

Plot the kNN (k-nearest neighbors) distance plot to visually detect the "elbow" and define an appropriate value for eps DBSCAN parameter.

Usage

```
plot_kNNdist(
  sessions,
  MinPts = NULL,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

sessions	tibble, sessions data set in evprof standard format.
MinPts	integer, DBSCAN MinPts parameter. If null, a value of 200 will be considered.
log	logical, whether to transform ConnectionStartDateTime and ConnectionHours variables to natural logarithmic scale (base = exp(1)).
start	integer, start hour in the x axis of the plot.

Details

The kNN (k-nearest neighbors) distance plot can provide insights into setting the eps parameter in DBSCAN. The "elbow" in the kNN distance plot is the point where the distances start to increase significantly. At the same time, for DBSCAN, the eps parameter defines the radius within which a specified number of points must exist for a data point to be considered a core point. Therefore, the "elbow" of the kNN distance plot can provide a sense of the scale of the data and help you choose a reasonable range for the eps parameter in DBSCAN.

Value

plot

Examples

```
library(dplyr)
california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_kNNdist(start = 3, log = TRUE)
```

---

plot_model_clusters *Plot all bi-variable GMM (clusters) with the colors corresponding to the assigned user profile. This shows which clusters correspond to which user profile, and the proportion of every user profile.*

Description

Plot all bi-variable GMM (clusters) with the colors corresponding to the assigned user profile. This shows which clusters correspond to which user profile, and the proportion of every user profile.

Usage

```
plot_model_clusters(
  subsets_clustering = list(),
  clusters_definition = list(),
  profiles_ratios,
  log = TRUE
)
```

Arguments

subsets_clustering	list with clustering results of each subset (direct output from function `cluser_sessions()`)
clusters_definition	list of tibbles with clusters definitions (direct output from function `define_clusters()`) of each sub-set
profiles_ratios	tibble with columns `profile` and `ratio`
log	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).

Value

ggplot2

Examples

```
library(dplyr)

# Select working day sessions (`Timecycle == 1`) that
# disconnect the same day (`Disconnection == 1`)
sessions_day <- evprof::california_ev_sessions_profiles %>%
  filter(Timecycle == "Workday") %>%
  sample_frac(0.05)
plot_points(sessions_day, start = 3)

# Identify two clusters
sessions_clusters <- cluster_sessions(
  sessions_day, k=2, seed = 1234, log = TRUE
)

# Plot the clusters found
plot_bivarGMM(
  sessions = sessions_clusters$sessions,
  models = sessions_clusters$models,
  log = TRUE, start = 3
)

# Define the clusters with user profile interpretations
clusters_definitions <- define_clusters(
  models = sessions_clusters$models,
```

```
  interpretations = c(
    "Connections during all day (high variability)",
    "Connections during working hours"#'
  ),
  profile_names = c("Visitors", "Workers"),
  log = TRUE
)

# Create a table with the connection GMM parameters
connection_models <- get_connection_models(
  subsets_clustering = list(sessions_clusters),
  clusters_definition = list(clusters_definitions)
)

# Plot all bi-variable GMM (clusters) with the colors corresponding
# to their assigned user profile
plot_model_clusters(
  subsets_clustering = list(sessions_clusters),
  clusters_definition = list(clusters_definitions),
  profiles_ratios = connection_models[c("profile", "ratio")]
)
```

`plot_outliers` *Plot outlying sessions*

Description

Plot outlying sessions

Usage

```
plot_outliers(
  sessions,
  start = getOption("evprof.start.hour"),
  log = FALSE,
  ...
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`start`	integer, start hour in the x axis of the plot.
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).
`...`	arguments to pass to function ggplot2::plot_point

Value

ggplot2 plot

Examples

```
library(dplyr)
sessions_outliers <- california_ev_sessions %>%
  sample_frac(0.05) %>%
  detect_outliers(start = 3, noise_th = 5, eps = 2.5)
plot_outliers(sessions_outliers, start = 3)
plot_outliers(sessions_outliers, start = 3, log = TRUE)
```

---

plot_points *Scatter plot of sessions*

Description

Scatter plot of sessions

Usage

```
plot_points(sessions, start = getOption("evprof.start.hour"), log = FALSE, ...)
```

Arguments

sessions	tibble, sessions data set in evprof standard format.
start	integer, start hour in the x axis of the plot.
log	logical, whether to transform ConnectionStartDateTime and ConnectionHours variables to natural logarithmic scale (base = exp(1)).
...	arguments to ggplot2::geom_point function

Value

ggplot scatter plot

Examples

```
library(dplyr)
california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_points()
california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_points(start = 3)
california_ev_sessions %>%
  sample_frac(0.05) %>%
  plot_points(log = TRUE)
```

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

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"))
```

round_to_interval *Round to nearest interval*

Description

Round to nearest interval

Usage

```
round_to_interval(dbl, interval)
```

Arguments

dbl	number to round
interval	rounding interval

Value

numeric value

Examples

```
set.seed(1)
random_vct <- rnorm(10, 5, 5)
round_to_interval(random_vct, 2.5)
```

`save_clustering_iterations` *Save iteration plots in PDF file*

Description

Save iteration plots in PDF file

Usage

```
save_clustering_iterations(
  sessions,
  k,
  filename,
  it = 12,
  seeds = round(runif(it, min = 1, max = 1000)),
  plot_scale = 2,
  points_size = 0.25,
  mclust_tol = 1e-08,
  mclust_itmax = 10000,
  log = FALSE,
  start = getOption("evprof.start.hour")
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format.
`k`	number of clusters
`filename`	string defining the PDF output file path (with extension .pdf)
`it`	number of iterations
`seeds`	seed for each iteration
`plot_scale`	scale of each iteration plot for a good visualization in pdf file
`points_size`	integer, size of points in the scatter plot
`mclust_tol`	tolerance parameter for clustering
`mclust_itmax`	maximum number of iterations
`log`	logical, whether to transform `ConnectionStartDateTime` and `ConnectionHours` variables to natural logarithmic scale (base = `exp(1)`).
`start`	integer, start hour in the x axis of the plot.

Value

nothing, but a PDF file is saved in the path specified by parameter `filename`

Examples

```
temp_file <- file.path(tempdir(), "iteration.pdf")
save_clustering_iterations(california_ev_sessions, k = 2, it = 4, filename = temp_file)
```

---

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

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"))
```

`set_profiles` *Classify sessions into user profiles*

Description

Joins all sub-sets from the list, adding a new column `Profile`

Usage

```
set_profiles(sessions_clustered = list(), clusters_definition = list())
```

Arguments

`sessions_clustered`
: list of tibbles with sessions clustered (`sessions`object of the output from function `cluser_sessions()`) from each sub-set

`clusters_definition`
: list of tibbles with clusters definitions (direct output from function `define_clusters()`) of each sub-set

Value

tibble

Examples

```
library(dplyr)

# Select working day sessions (`Timecycle == 1`) that
# disconnect the same day (`Disconnection == 1`)
sessions_day <- california_ev_sessions %>%
  divide_by_timecycle(
    months_cycles = list(1:12), # Not differentiation between months
    wdays_cycles = list(1:5, 6:7) # Differentiation between workdays/weekends
  ) %>%
  divide_by_disconnection(
    division_hour = 10, start = 3
  ) %>%
  filter(
    Disconnection == 1, Timecycle == 1
  ) %>%
  sample_frac(0.05)

# Identify two clusters
sessions_clusters <- cluster_sessions(
  sessions_day, k=2, seed = 1234, log = TRUE
)

# Plot the clusters found
```

```
plot_bivarGMM(
  sessions = sessions_clusters$sessions,
  models = sessions_clusters$models,
  log = TRUE, start = 3
)

# Define the clusters with user profile interpretations
clusters_definitions <- define_clusters(
  models = sessions_clusters$models,
  interpretations = c(
    "Connections during working hours",
    "Connections during all day (high variability)"
  ),
  profile_names = c("Workers", "Visitors"),
  log = TRUE
)

# Classify each session to the corresponding user profile
sessions_profiles <- set_profiles(
  sessions_clustered = list(sessions_clusters$sessions),
  clusters_definition = list(clusters_definitions)
)
```

`summarise_sessions` *Statistic summary of sessions features*

Description

Statistic summary of sessions features

Usage

```
summarise_sessions(
  sessions,
  .funs,
  vars = evprof::sessions_summary_feature_names
)
```

Arguments

`sessions`	tibble, sessions data set in evprof standard format. standard format.
`.funs`	A function to compute, e.g. `mean`, `max`, etc.
`vars`	character vector, variables to compute the histogram for

Value

Summary table

Examples

```
summarise_sessions(california_ev_sessions, mean)
```

Index