Package: epishiny (via r-universe)

October 24, 2024

Title Tools for interactive visualisation of epidemiological data

Version 0.0.0.9000

Description Time, place and person analysis using the R shiny web-framework.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports cli, shiny (>= 1.5.0), bslib (>= 0.5.1), bsicons, htmlwidgets, shinyWidgets, shinyjs, waiter, rlang, magrittr, tibble, dplyr, tidyr, purrr, stringr, forcats, scales, glue, lubridate, highcharter, gt, gtsummary, sf, leaflet, leaflet.minicharts, chromote, webshot2

Depends R (>= 2.10)

URL `https://github.com/epicentre-msf/epishiny`, `https://epicentre-msf.github.io/epishiny/`

BugReports `https://github.com/epicentre-msf/epishiny/issues`

Suggests knitr, rmarkdown, rnaturalearth, readr

VignetteBuilder knitr

Repository https://epicentre-msf.r-universe.dev

RemoteUrl https://github.com/epicentre-msf/epishiny

RemoteRef HEAD

RemoteSha 88a4820ee1667e38b5881e73718a7386a85b48b0

Contents

`df_ll` *Example Linelist Data*

Description

A ’linelist’ is a (tidy) data format used in public health data collection with each row representing an individual (patient, participant, etc) and each column representing a variable associated with said individual.

Usage

```
df_ll
```

Format

a tibble dataframe

Details

`df_ll` is an example linelist dataset containing data for a fake measles outbreak in Yemen. The data contains temporal, demographic, and geographic information for each patient, as well as other medical indicators.

Examples

```
df_ll
```

filter_ui *Filter module*

Description

Filter linelist data using a sidebar with shiny inputs.

Usage

```
filter_ui(
  id,
  group_vars,
  date_range,
  title = NULL,
  date_filters_lab = "Date filters",
  period_lab = "Period",
  missing_dates_lab = "Include patients with missing dates?",
  group_filters_lab = "Group filters",
  filter_btn_lab = "Filter",
  reset_btn_lab = "Reset"
)

filter_server(
  id,
  df,
  date_var,
  group_vars,
  time_filter = shiny::reactiveVal(),
  place_filter = shiny::reactiveVal(),
  na_label = getOption("epishiny.na.label", "(Missing)")
)
```

Arguments

`id`	Module id. Must be the same in both the UI and server function to link the two.
`group_vars`	named character vector of categorical variables for the data grouping input. Names are used as variable labels.
`date_range`	A vector containing the minimum and maximum dates for the date range input.
`title`	The title of the sidebar.
`date_filters_lab`	The label for the date filters accordion panel.
`period_lab`	The label for the date range input.
`missing_dates_lab`	The label for the include missing dates checkbox.
`group_filters_lab`	The label for the group filters accordion panel.

`filter_btn_lab`	The label for the filter data button.
`reset_btn_lab`	The label for the reset filters button.
`df`	Data frame or tibble of patient level or aggregated data. Can be either a shiny reactive or static dataset.
`date_var`	The name of the date variable in the data frame to be filtered on.
`time_filter`	supply the output of `time_server()` wrapped in a `shiny::reactive()` here to add its filter information to the filter sidebar
`place_filter`	supply the output of `place_server()` wrapped in a `shiny::reactive()` here to add its filter information to the filter sidebar
`na_label`	The label to use for missing values in group variables.

Value

A bslib::sidebar UI element with date filters, group filters, and action buttons.

The server function returns both the filtered data and a formatted text string with filter information named `df` and `filter_info` respectively in a reactive list. These should be passed as arguments of the same name in the time, place and person modules wrapped in a `shiny::reactive()`

Examples

```
library(shiny)
library(bslib)
library(epishiny)

# example package data
data("df_ll") # linelist
data("sf_yem") # sf geo boundaries for Yemen admin 1 & 2

# setup geo data for adm1 and adm2 using the
# geo_layer function to be passed to the place module
# if population variable is provided, attack rates
# will be shown on the map as a choropleth
geo_data <- list(
  geo_layer(
    layer_name = "Governorate", # name of the boundary level
    sf = sf_yem$adm1, # sf object with boundary polygons
    name_var = "adm1_name", # column with place names
    pop_var = "adm1_pop", # column with population data (optional)
    join_by = c("pcode" = "adm1_pcode") # geo to data join vars: LHS = sf, RHS = data
  ),
  geo_layer(
    layer_name = "District",
    sf = sf_yem$adm2,
    name_var = "adm2_name",
    pop_var = "adm2_pop",
    join_by = c("pcode" = "adm2_pcode")
  )
)
```

```
# range of dates used in filter module to filter time period
date_range <- range(df_ll$date_notification, na.rm = TRUE)

# define date variables in data as named list to be used in app
date_vars <- c(
  "Date of notification" = "date_notification",
  "Date of onset" = "date_symptom_start",
  "Date of hospitalisation" = "date_hospitalisation_start",
  "Date of outcome" = "date_hospitalisation_end"
)

# define categorical grouping variables
# in data as named list to be used in app
group_vars <- c(
  "Governorate" = "adm1_origin",
  "Sex" = "sex_id",
  "Hospitalised" = "hospitalised_yn",
  "Vaccinated measles" = "vacci_measles_yn",
  "Outcome" = "outcome"
)

# user interface
ui <- page_sidebar(
  title = "epishiny",
  # sidebar
  sidebar = filter_ui(
    "filter",
    group_vars = group_vars,
    date_range = date_range,
    period_lab = "Notification period"
  ),
  # main content
  layout_columns(
    col_widths = c(12, 7, 5),
    place_ui(
      id = "map",
      geo_data = geo_data,
      group_vars = group_vars
    ),
    time_ui(
      id = "curve",
      title = "Time",
      date_vars = date_vars,
      group_vars = group_vars,
      ratio_line_lab = "Show CFR line?"
    ),
    person_ui(id = "age_sex")
  )
)

# app server
server <- function(input, output, session) {
  app_data <- filter_server(
    id = "filter",
    df = df_ll,
    date_var = "date_notification",
    group_vars = group_vars
  )
  place_server(
    id = "map",
    df = reactive(app_data()$df),
    geo_data = geo_data,
    group_vars = group_vars,
    filter_info = reactive(app_data()$filter_info)
  )
  time_server(
    id = "curve",
    df = reactive(app_data()$df),
    date_vars = date_vars,
    group_vars = group_vars,
    show_ratio = TRUE,
    ratio_var = "outcome",
    ratio_lab = "CFR",
    ratio_numer = "Deceased",
    ratio_denom = c("Deceased", "Healed", "Abandonment"),
    filter_info = reactive(app_data()$filter_info)
  )
  person_server(
    id = "age_sex",
    df = reactive(app_data()$df),
    age_var = "age_years",
    sex_var = "sex_id",
    male_level = "Male",
    female_level = "Female",
    filter_info = reactive(app_data()$filter_info)
  )
}

# launch app
if (interactive()) {
  shinyApp(ui, server)
}
```

geo_layer *Build a geo layer to be used in the 'place' module*

Description

Build a geo layer to be used in the 'place' module

Usage

```
geo_layer(layer_name, sf, name_var, join_by, pop_var = NULL)
```

Arguments

`layer_name`	the name of the geo layer, for example 'State', 'Department', 'Admin2' etc. If providing multiple layers, layer names must be unique.
`sf`	geographical data of class 'sf' (simple features).
`name_var`	character string of the variable name in `sf` containing the names of each geographical feature.
`join_by`	data join specification to join geo layer to a dataset. Should be either a single variable name present in both datasets or a named vector where the name is the geo layer join variable and the value is the join variable of the dataset. i.e. `c("pcode" = "place_code")` LHS = geo, RHS = data.
`pop_var`	character string of the variable name in `sf` containing population data for each feature. If provided, attack rates will be shown on the map as a choropleth.

Value

named list of class "epishiny_geo_layer"

Examples

```
geo_layer(
  layer_name = "Governorate",
  sf = sf_yem$adm1,
  name_var = "adm1_name",
  pop_var = "adm1_pop",
  join_by = c("pcode" = "adm1_pcode")
)
```

`launch_demo_dashboard` *Launch epishiny demo dashboard*

Description

See an example of the type of dashboard you can build using `epishiny` modules within a `bslib` UI.

Usage

```
launch_demo_dashboard()
```

Value

No return value, a shiny app is launched.

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  library(epishiny)
  launch_demo_dashboard()
}
```

`launch_module` *Launch a single 'epishiny' module as a standalone shiny app*

Description

Use this function to quickly launch any of the 3 'epishiny' interactive visualisation modules (time, place, person) independently, allowing for incorporation into exploratory data analysis workflows in R.

Usage

```
launch_module(module = c("time", "place", "person"), ...)
```

Arguments

`module`	Name of the module to launch. Current options are "time", "place" or "person".
`...`	Other named arguments passed to the relevant module UI and Server functions. See each module's documentation for details of the arguments required.

Value

No return value, a shiny app is launched.

Examples

```
library(shiny)
library(epishiny)

# example package data
data("df_ll")
data("sf_yem")

# setup geo data for adm1 and adm2 using the
# geo_layer function to be passed to the place module
# if population variable is provided, attack rates
# will be shown on the map as a choropleth
geo_data <- list(
  geo_layer(
    layer_name = "Governorate", # name of the boundary level
    sf = sf_yem$adm1, # sf object with boundary polygons
    name_var = "adm1_name", # column with place names
    pop_var = "adm1_pop", # column with population data (optional)
    join_by = c("pcode" = "adm1_pcode") # geo to data join vars: LHS = sf, RHS = data
  ),
  geo_layer(
    layer_name = "District",
    sf = sf_yem$adm2,
    name_var = "adm2_name",
    pop_var = "adm2_pop",
    join_by = c("pcode" = "adm2_pcode")
  )
)

# define date variables in data as named list to be used in app
date_vars <- c(
  "Date of notification" = "date_notification",
  "Date of onset" = "date_symptom_start",
  "Date of hospitalisation" = "date_hospitalisation_start",
  "Date of outcome" = "date_hospitalisation_end"
)

# define categorical grouping variables
# in data as named list to be used in app
group_vars <- c(
  "Governorate" = "adm1_origin",
  "Sex" = "sex_id",
  "Hospitalised" = "hospitalised_yn",
  "Vaccinated measles" = "vacci_measles_yn",
  "Outcome" = "outcome"
)

# launch time epicurve module
if (interactive()) {
  launch_module(
    module = "time",
    df = df_ll,
    date_vars = date_vars,
    group_vars = group_vars,
    show_ratio = TRUE,
    ratio_line_lab = "Show CFR line?",
    ratio_var = "outcome",
    ratio_lab = "CFR",
    ratio_numer = "Deceased",
    ratio_denom = c("Deceased", "Healed", "Abandonment")
  )
}

# launch place map module
if (interactive()) {
  launch_module(
    module = "place",
    df = df_ll,
    geo_data = geo_data,
    group_vars = group_vars
  )
}

# launch person age/sex pyramid module
if (interactive()) {
  launch_module(
    module = "person",
    df = df_ll,
    age_var = "age_years",
    sex_var = "sex_id",
    male_level = "Male",
    female_level = "Female"
  )
}
```

`person_ui` *Person module*

Description

Visualise age and sex demographics in a population pyramid chart and summary table.

Usage

```
person_ui(
  id,
  count_vars = NULL,
  title = "Person",
  icon = bsicons::bs_icon("people-fill"),
  opts_btn_lab = "options",
  count_vars_lab = "Indicator",
  full_screen = TRUE
)

person_server(
  id,
  df,
  sex_var,
  male_level,
  female_level,
  age_group_var = NULL,
  age_var = NULL,
  count_vars = NULL,
  age_breaks = c(0, 5, 18, 25, 35, 50, Inf),
  age_labels = c("<5", "5-17", "18-24", "25-34", "35-49", "50+"),
  age_var_lab = "Age (years)",
  age_group_lab = "Age group",
  n_lab = "N patients",
  colours = c("#19a0aa", "#f15f36"),
  filter_info = shiny::reactiveVal(),
  time_filter = shiny::reactiveVal(),
  place_filter = shiny::reactiveVal()
)
```

Arguments

`id`	Module id. Must be the same in both the UI and server function to link the two.
`count_vars`	If data is aggregated, variable name(s) of count variable(s) in data. If more than one variable provided, a select input will appear in the options dropdown. If named, names are used as variable labels.
`title`	The title for the card.
`icon`	The icon to display next to the title.
`opts_btn_lab`	The label for the options button.
`count_vars_lab`	text label for the aggregate count variables input.
`full_screen`	Add button to card to with the option to enter full screen mode?
`df`	Data frame or tibble of patient level or aggregated data. Can be either a shiny reactive or static dataset.
`sex_var`	The name of the sex variable in the data.
`male_level`	The level representing males in the sex variable.
`female_level`	The level representing females in the sex variable.
`age_group_var`	The name of a character/factor variable in the data with age groups. If specified, `age_var` is ignored.
`age_var`	The name of a numeric age variable in the data. If ages have already been binned into groups, use `age_group_var` instead.
`age_breaks`	A numeric vector specifying age breaks for age groups.
`age_labels`	Labels corresponding to the age breaks.
`age_var_lab`	The label for the age variable.
`age_group_lab`	The label for the age group variable.
`n_lab`	The label for the raw count variable.
`colours`	Vector of 2 colours to represent male and female, respectively.
`filter_info`	If contained within an app using `filter_server()`, supply the `filter_info` object returned by that function here wrapped in a `shiny::reactive()` to add filter information to chart exports.
`time_filter`	supply the output of `time_server()` wrapped in a `shiny::reactive()` here to filter the data by click events on the time module bar chart (clicking a bar will filter the data to the period the bar represents)
`place_filter`	supply the output of `place_server()` wrapped in a `shiny::reactive()` here to filter the data by click events on the place module map (clicking a polygon will filter the data to the clicked region)

Value

A bslib::navset_card_tab UI element with chart and table tabs.

Examples

```
library(shiny)
library(bslib)
library(epishiny)

# example package data
data("df_ll") # linelist
data("sf_yem") # sf geo boundaries for Yemen admin 1 & 2

# setup geo data for adm1 and adm2 using the
# geo_layer function to be passed to the place module
# if population variable is provided, attack rates
# will be shown on the map as a choropleth
geo_data <- list(
  geo_layer(
    layer_name = "Governorate", # name of the boundary level
    sf = sf_yem$adm1, # sf object with boundary polygons
    name_var = "adm1_name", # column with place names
    pop_var = "adm1_pop", # column with population data (optional)
    join_by = c("pcode" = "adm1_pcode") # geo to data join vars: LHS = sf, RHS = data
  ),
  geo_layer(
    layer_name = "District",
    sf = sf_yem$adm2,
    name_var = "adm2_name",
    pop_var = "adm2_pop",
    join_by = c("pcode" = "adm2_pcode")
  )
)

# range of dates used in filter module to filter time period
date_range <- range(df_ll$date_notification, na.rm = TRUE)

# define date variables in data as named list to be used in app
date_vars <- c(
  "Date of notification" = "date_notification",
  "Date of onset" = "date_symptom_start",
  "Date of hospitalisation" = "date_hospitalisation_start",
  "Date of outcome" = "date_hospitalisation_end"
)

# define categorical grouping variables
# in data as named list to be used in app
group_vars <- c(
  "Governorate" = "adm1_origin",
  "Sex" = "sex_id",
  "Hospitalised" = "hospitalised_yn",
  "Vaccinated measles" = "vacci_measles_yn",
  "Outcome" = "outcome"
)

# user interface
```

```
ui <- page_sidebar(
  title = "epishiny",
  # sidebar
  sidebar = filter_ui(
    "filter",
    group_vars = group_vars,
    date_range = date_range,
    period_lab = "Notification period"
  ),
  # main content
  layout_columns(
    col_widths = c(12, 7, 5),
    place_ui(
      id = "map",
      geo_data = geo_data,
      group_vars = group_vars
    ),
    time_ui(
      id = "curve",
      title = "Time",
      date_vars = date_vars,
      group_vars = group_vars,
      ratio_line_lab = "Show CFR line?"
    ),
    person_ui(id = "age_sex")
  )
)

# app server
server <- function(input, output, session) {
  app_data <- filter_server(
    id = "filter",
    df = df_ll,
    date_var = "date_notification",
    group_vars = group_vars
  )
  place_server(
    id = "map",
    df = reactive(app_data()$df),
    geo_data = geo_data,
    group_vars = group_vars,
    filter_info = reactive(app_data()$filter_info)
  )
  time_server(
    id = "curve",
    df = reactive(app_data()$df),
    date_vars = date_vars,
    group_vars = group_vars,
    show_ratio = TRUE,
    ratio_var = "outcome",
    ratio_lab = "CFR",
    ratio_numer = "Deceased",
    ratio_denom = c("Deceased", "Healed", "Abandonment"),
    filter_info = reactive(app_data()$filter_info)
  )
  person_server(
    id = "age_sex",
    df = reactive(app_data()$df),
    age_var = "age_years",
    sex_var = "sex_id",
    male_level = "Male",
    female_level = "Female",
    filter_info = reactive(app_data()$filter_info)
  )
}

# launch app
if (interactive()) {
  shinyApp(ui, server)
}
```

`place_ui` *Place module*

Description

Visualise geographical distribution across multiple administrative boundaries on an interactive leaflet map.

Usage

```
place_ui(
  id,
  geo_data,
  count_vars = NULL,
  group_vars = NULL,
  title = "Place",
  icon = bsicons::bs_icon("geo-fill"),
  tooltip = NULL,
  geo_lab = "Geo boundaries",
  count_vars_lab = "Indicator",
  groups_lab = "Group data by",
  no_grouping_lab = "No grouping",
  circle_size_lab = "Circle size multiplyer",
  opts_btn_lab = "options",
  download_lab = "download",
  full_screen = TRUE
)

place_server(
  id,
  df,
  geo_data,
  count_vars = NULL,
  group_vars = NULL,
  show_parent_borders = FALSE,
  choro_lab = "Rate /100 000",
  choro_pal = "Reds",
  choro_opacity = 0.7,
  export_width = 1200,
  export_height = 650,
  time_filter = shiny::reactiveVal(),
  filter_info = shiny::reactiveVal(),
  filter_reset = shiny::reactiveVal()
)
```

Arguments

`id`	Module id. Must be the same in both the UI and server function to link the two.
`geo_data`	A list of named lists containing spatial sf dataframes and other information for different geographical levels.
`count_vars`	If data is aggregated, variable name(s) of count variable(s) in data. If more than one is variable provided, a select input will appear in the options dropdown. If named, names are used as variable labels.
`group_vars`	Character vector of categorical variable names. If provided, a select input will appear in the options dropdown allowing for data groups to be visualised on the map in pie charts per geographical unit. If named, names are used as variable labels.
`title`	The title for the card.
`icon`	The icon to be displayed next to the title
`tooltip`	additional title hover text information
`geo_lab`	The label for the geographical level selection.
`count_vars_lab`	text label for the aggregate count variables input.
`groups_lab`	The label for the group data by selection.
`no_grouping_lab`	text label for the no grouping option in the grouping input.
`circle_size_lab`	text label for the circle size slider input.
`opts_btn_lab`	text label for the dropdown menu button.
`download_lab`	text label for the download button.
`full_screen`	Add button to card to with the option to enter full screen mode?
`df`	Data frame or tibble of patient level or aggregated data. Can be either a shiny reactive or static dataset.
`show_parent_borders`	Show borders of parent boundary levels?

`choro_lab`	Label for attack rate choropleth (only applicable if `geo_data` contains population data)
`choro_pal`	Colour palette passed to `leaflet::colorBin()` for attack rate choropleth (only applicable if `geo_data` contains population data)
`choro_opacity`	Opacity of choropleth colour (only applicable if `geo_data` contains population data)
`export_width`	The width of the exported map image.
`export_height`	The height of the exported map image.
`time_filter`	supply the output of `time_server()` wrapped in a `shiny::reactive()` here to filter the data by click events on the time module bar chart (clicking a bar will filter the data to the period the bar represents)
`filter_info`	If contained within an app using `filter_server()`, supply the `filter_info` object returned by that function here wrapped in a `shiny::reactive()` to add filter information to chart exports.
`filter_reset`	If contained within an app using `filter_server()`, supply the `filter_reset` object returned by that function here wrapped in a `shiny::reactive()` to reset any click event filters that have been set from by module.

Value

A bslib::card UI element with options and download button and a leaflet map.

The server function returns the leaflet map's shape click information as a list.

Examples

```
library(shiny)
library(bslib)
library(epishiny)

# example package data
data("df_ll") # linelist
data("sf_yem") # sf geo boundaries for Yemen admin 1 & 2

# setup geo data for adm1 and adm2 using the
# geo_layer function to be passed to the place module
# if population variable is provided, attack rates
# will be shown on the map as a choropleth
geo_data <- list(
  geo_layer(
    layer_name = "Governorate", # name of the boundary level
    sf = sf_yem$adm1, # sf object with boundary polygons
    name_var = "adm1_name", # column with place names
    pop_var = "adm1_pop", # column with population data (optional)
    join_by = c("pcode" = "adm1_pcode") # geo to data join vars: LHS = sf, RHS = data
  ),
  geo_layer(
    layer_name = "District",
    sf = sf_yem$adm2,
    name_var = "adm2_name",
    pop_var = "adm2_pop",
    join_by = c("pcode" = "adm2_pcode")
  )
)

# range of dates used in filter module to filter time period
date_range <- range(df_ll$date_notification, na.rm = TRUE)

# define date variables in data as named list to be used in app
date_vars <- c(
  "Date of notification" = "date_notification",
  "Date of onset" = "date_symptom_start",
  "Date of hospitalisation" = "date_hospitalisation_start",
  "Date of outcome" = "date_hospitalisation_end"
)

# define categorical grouping variables
# in data as named list to be used in app
group_vars <- c(
  "Governorate" = "adm1_origin",
  "Sex" = "sex_id",
  "Hospitalised" = "hospitalised_yn",
  "Vaccinated measles" = "vacci_measles_yn",
  "Outcome" = "outcome"
)

# user interface
ui <- page_sidebar(
  title = "epishiny",
  # sidebar
  sidebar = filter_ui(
    "filter",
    group_vars = group_vars,
    date_range = date_range,
    period_lab = "Notification period"
  ),
  # main content
  layout_columns(
    col_widths = c(12, 7, 5),
    place_ui(
      id = "map",
      geo_data = geo_data,
      group_vars = group_vars
    ),
    time_ui(
      id = "curve",
      title = "Time",
      date_vars = date_vars,
      group_vars = group_vars,
      ratio_line_lab = "Show CFR line?"
    ),
    person_ui(id = "age_sex")
  )
)

# app server
server <- function(input, output, session) {
  app_data <- filter_server(
    id = "filter",
    df = df_ll,
    date_var = "date_notification",
    group_vars = group_vars
  )
  place_server(
    id = "map",
    df = reactive(app_data()$df),
    geo_data = geo_data,
    group_vars = group_vars,
    filter_info = reactive(app_data()$filter_info)
  )
  time_server(
    id = "curve",
    df = reactive(app_data()$df),
    date_vars = date_vars,
    group_vars = group_vars,
    show_ratio = TRUE,
    ratio_var = "outcome",
    ratio_lab = "CFR",
    ratio_numer = "Deceased",
    ratio_denom = c("Deceased", "Healed", "Abandonment"),
    filter_info = reactive(app_data()$filter_info)
  )
  person_server(
    id = "age_sex",
    df = reactive(app_data()$df),
    age_var = "age_years",
    sex_var = "sex_id",
    male_level = "Male",
    female_level = "Female",
    filter_info = reactive(app_data()$filter_info)
  )
}

# launch app
if (interactive()) {
  shinyApp(ui, server)
}
```

sf_yem *Yemen Governorate (adm1) and District (adm2) Administrative Boundaries*

Description

A list of length 2 containing geographic administrative boundary data for Yemen, stored as simple features (sf) objects.

Usage

```
sf_yem
```

Format

named list of sf objects

Details

Each admin level can be joined to the example `df_ll` dataset with a join by specification of `c("pcode" = "adm1_pcode")` and `c("pcode" = "adm2_pcode")` respectively. These should be passed as the `join_by` field in each `geo_data` specification passed to place_ui and place_server.

Examples

```
sf_yem$adm1
sf_yem$adm2
```

`time_ui` *Time module*

Description

Visualise data over time with an interactive 'epicurve'.

Usage

```
time_ui(
  id,
  date_vars,
  count_vars = NULL,
  group_vars = NULL,
  title = "Time",
  icon = bsicons::bs_icon("bar-chart-line-fill"),
  tooltip = NULL,
  opts_btn_lab = "options",
  date_lab = "Date axis",
  date_int_lab = "Date interval",
  date_intervals = c(Day = "day", Week = "week", Month = "month"),
  count_vars_lab = "Indicator",
  groups_lab = "Group data by",
  no_grouping_lab = "No grouping",
  bar_stacking_lab = "Bar stacking",
  cumul_data_lab = "Show cumulative data?",
  ratio_line_lab = "Show ratio line?",
  full_screen = TRUE
)

time_server(
  id,
  df,
  date_vars,
  count_vars = NULL,
  group_vars = NULL,
  show_ratio = FALSE,
  ratio_var = NULL,
  ratio_lab = NULL,
  ratio_numer = NULL,
  ratio_denom = NULL,
  place_filter = shiny::reactiveVal(),
  filter_info = shiny::reactiveVal(),
  filter_reset = shiny::reactiveVal()
)
```

Arguments

`id`	Module id. Must be the same in both the UI and server function to link the two.
`date_vars`	Character vector of date variable(s) for the date axis. If named, names are used as variable labels.
`count_vars`	If data is aggregated, variable name(s) of count variable(s) in data. If more than one variable provided, a select input will appear in the options dropdown. If named, names are used as variable labels.
`group_vars`	Character vector of categorical variable names. If provided, a select input will appear in the options dropdown allowing for data groups to be visualised as stacked bars on the epicurve. If named, names are used as variable labels.
`title`	Header title for the card.
`icon`	The icon to display next to the title.
`tooltip`	additional title hover text information
`opts_btn_lab`	text label for the dropdown menu button.
`date_lab`	text label for the date variable input.
`date_int_lab`	text label for the date interval input.
`date_intervals`	Character vector with choices for date aggregation intervals passed to the `unit` argument of lubridate::floor_date. If named, names are used as labels. Default is c('day', 'week', 'year').
`count_vars_lab`	text label for the aggregate count variables input.
`groups_lab`	text label for the grouping variable input.

`no_grouping_lab`	text label for the no grouping option in the grouping input.
`bar_stacking_lab`	text label for bar stacking option.
`cumul_data_lab`	text label for cumulative data option.
`ratio_line_lab`	text label for the ratio line input. This input will only be visable if `show_ratio` is TRUE in time_server
`full_screen`	Add button to card to with the option to enter full screen mode?
`df`	Data frame or tibble of patient level or aggregated data. Can be either a shiny reactive or static dataset.
`show_ratio`	Display a ratio line on the epicurve?
`ratio_var`	For patient level data, character string of variable name to use for ratio calculation.
`ratio_lab`	The label to describe the computed ratio i.e. 'CFR' for case fatality ratio.
`ratio_numer`	For patient level data, Value(s) in `ratio_var` to be used for the ratio numerator i.e. 'Death'. For aggregated data, character string of numeric count column to use of ratio numerator i.e. 'deaths'.
`ratio_denom`	For patient level data, values in `ratio_var` to be used for the ratio denominator i.e. `c('Death', 'Recovery')`. For aggregated data, character string of numeric count column to use of ratio denominator i.e. 'cases'.
`place_filter`	supply the output of `place_server()` wrapped in a `shiny::reactive()` here to filter the data by click events on the place module map (clicking a polygon will filter the data to the clicked region)
`filter_info`	If contained within an app using `filter_server()`, supply the `filter_info` object returned by that function here wrapped in a `shiny::reactive()` to add filter information to chart exports.
`filter_reset`	If contained within an app using `filter_server()`, supply the `filter_reset` object returned by that function here wrapped in a `shiny::reactive()` to reset any click event filters that have been set from by module.

Value

the module server function returns any point click event data of the highchart. see highcharter::hc_add_event_point for details.

Index