Package ‘trelliscopejs’

October 14, 2022

Title  Create Interactive Trelliscope Displays

Version 0.2.6

Description  Trelliscope is a scalable, flexible, interactive approach to visualizing data (Hafen, 2013 <doi:10.1109/LDAV.2013.6675164>). This package provides methods that make it easy to create a Trelliscope display specification for TrelliscopeJS. High-level functions are provided for creating displays from within 'tidyverse' or 'ggplot2' workflows. Low-level functions are also provided for creating new interfaces.

Depends  R (>= 3.4.0)

License BSD_3_clause + file LICENSE

Encoding UTF-8

LazyData true

Imports dplyr, purrr, grid, htmltools, DistributionUtils, grDevices, gtable, digest, jsonlite, ggplot2 (>= 3.2.1), base64enc, htmlwidgets, graphics, progress, utils, knitr, webshot, autocogs, tidyr, rlang

Suggests plotly, testthat, covr, gapminder, rmarkdown, shiny, housingData

RoxygenNote 7.1.1

URL https://github.com/hafen/trelliscopejs

BugReports https://github.com/hafen/trelliscopejs/issues

VignetteBuilder knitr

NeedsCompilation no

Author Ryan Hafen [aut, cre] (<https://orcid.org/0000-0002-5516-8367>), Barret Schloerke [aut]

Maintainer Ryan Hafen <rhafen@gmail.com>

Repository CRAN

Date/Publication 2021-02-01 08:00:02 UTC
Description

Create interactive Trelliscope displays

Details

https://hafen.github.io/trelliscopejs/

Examples

help(package = trelliscopejs)
as_cognostics  

**Cast a data frame as a cognostics data frame**

**Description**

Cast a data frame as a cognostics data frame

**Usage**

```r
as_cognostics(
  x,
  cond_cols,
  key_col = NULL,
  cog_desc = NULL,
  needs_key = TRUE,
  needsCond = TRUE,
  group = "common"
)
```

**Arguments**

- `x` a data frame
- `cond_cols` the column name(s) that comprise the conditioning variables
- `key_col` the column name that indicates the panel key
- `cog_desc` an optional named list of descriptions for the cognostics columns
- `needs_key` does the result need to have a "key" column?
- `needsCond` does the result need to have conditioning variable columns?
- `group` value to be used in the `cog` group

---

cog  

**Cast Column as a Cognostic**

**Description**

Cast a column of a cognostics data frame as a cognostic object
Usage

cog(
  val = NULL,
  desc = "",
  group = "common",
  type = NULL,
  default_label = FALSE,
  default_active = TRUE,
  filterable = TRUE,
  sortable = TRUE,
  log = NULL
)

Arguments

val a scalar value (numeric, character, date, etc.)
desc a description for this cognostic value
group optional categorization of the cognostic for organizational purposes in the viewer (currently not implemented in the viewer)
type the desired type of cognostic you would like to compute (see details)
default_label should this cognostic be used as a panel label in the viewer by default?
default_active should this cognostic be active (available for sort / filter / sample) by default?
filterable should this cognostic be filterable? Default is TRUE. It can be useful to set this to FALSE if the cognostic is categorical with many unique values and is only desired to be used as a panel label.
sortable should this cognostic be sortable?
log when being used in the viewer for visual univariate and bivariate filters, should the log be computed? Useful when the distribution of the cognostic is very long-tailed or has large outliers. Can either be a logical or a positive integer indicating the base.

Details

Different types of cognostics can be specified through the type argument that will affect how the user is able to interact with those cognostics in the viewer. This can usually be ignored because it will be inferred from the implicit data type of val. But there are special types of cognostics, such as geographic coordinates and relations (not implemented) that can be specified as well. Current possibilities for type are "key", "integer", "numeric", "factor", "date", "time", "href".

Value

object of class "cog"
cogs

Examples

library(dplyr)
library(tidyr)
library(purrr)
library(ggplot2)
library(plotly)

mpg_cog <- mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(
    cogs = map_cog(data, ~ tibble(
      mean_city_mpg = cog(mean(.$cty), desc = "Mean city mpg"),
      mean_hwy_mpg = cog(mean(.$hwy), desc = "Mean highway mpg"),
      most_common_driv = cog(tail(names(table(.$drv)), 1), desc = "Most common drive type")
    ),
    panel = map_plot(data, function(x) {
      plot_ly(data = x, x = ~cty, y = ~hwy,
        type = "scatter", mode = "markers") %>%
      layout(
        xaxis = list(range = c(9, 47)),
        yaxis = list(range = c(7, 37))
      )
    })),
  
  trelliscope(mpg_cog, name = "city_vs_highway_mpg", nrow = 1, ncol = 2)

cogs

Cogs Wrapper Function

Description

Cogs Wrapper Function

Usage

cogs(.x, .f, ...)

Arguments

.x a list or atomic vector (see map for details)
.f a function, formula, or atomic vector (see map for details)
... additional arguments passed on to .f (see map for details)

Details

See map
Examples

```
library(dplyr)
library(tidyr)
library(plotly)
ggplot2::mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(
    additional_cogs = map_cog(data, function(x) {
      tibble(
        max_city_mpg = cog(max(x$cty), desc = "Max city mpg"),
        min_city_mpg = cog(min(x$cty), desc = "Min city mpg")
      )
    }),
    panel = map_plot(data, function(x) {
      plot_ly(data = x, x = ~cty, y = ~hwy,
              type = "scatter", mode = "markers")
    })
  ) %>%
  trelliscope(name = "city_vs_highway_mpg", nrow = 1, ncol = 2)
```

---

cog_disp_filter  

Helper function for creating a cognostic for a link to another display in a filtered state

Description

Helper function for creating a cognostic for a link to another display in a filtered state

Usage

```
cog_disp_filter(
  display, 
  var, 
  val, 
  desc = "link", 
  group = "common", 
  default_label = FALSE, 
  default_active = FALSE, 
  filterable = FALSE, 
  sortable = FALSE
)
```

Arguments

display  
A string indicating the name of the display to link to.

var  
A string indicating the variable name to filter on.

val  
A string indicating the value of the filter.
cog_href

<table>
<thead>
<tr>
<th>desc</th>
<th>a description for this cognostic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>optional categorization of the cognostic for organizational purposes in the viewer (currently not implemented in the viewer)</td>
</tr>
<tr>
<td>default_label</td>
<td>should this cognostic be used as a panel label in the viewer by default?</td>
</tr>
<tr>
<td>default_active</td>
<td>should this cognostic be active (available for sort / filter / sample) by default?</td>
</tr>
<tr>
<td>filterable</td>
<td>should this cognostic be filterable? Default is TRUE. It can be useful to set this to FALSE if the cognostic is categorical with many unique values and is only desired to be used as a panel label.</td>
</tr>
<tr>
<td>sortable</td>
<td>should this cognostic be sortable?</td>
</tr>
</tbody>
</table>

---

cog_href  
Href Cognostic

Description

Create href to be used as cognostics in a trelliscope display

Usage

cog_href(
  x,
  desc = "link",
  group = "common",
  default_label = FALSE,
  default_active = FALSE,
  filterable = FALSE,
  sortable = FALSE,
  log = FALSE
)

Arguments

  x  
  URL to link to
  desc, group, default_label, default_active, filterable, sortable, log
  arguments passed to cog

See Also

cog
Examples

library(dplyr)
library(tidyr)
library(plotly)
iris %>%
nest(data = ~Species) %>%
mutate(
  panel = map_plot(data, function(x) {
    plot_ly(data = x, x = ~Sepal.Length, y = ~Sepal.Width,
              type = "scatter", mode = "markers")
  }),
  wiki_link = cog_href(paste0("https://en.wikipedia.org/wiki/Iris_",
                              tolower(Species))[[1]], default_label = TRUE,
                         desc = "link to species on wikipedia")
) %>%
trelliscope(name = "iris_species", ncol = 3)

---

facet_trelliscope  Facet Trelliscope

Description

Facet Trelliscope

Usage

facet_trelliscope(
  facets,
  nrow = 1,
  ncol = 1,
  scales = "same",
  name = NULL,
  group = "common",
  desc = ggplot2::waiver(),
  md_desc = ggplot2::waiver(),
  path = NULL,
  height = 500,
  width = 500,
  state = NULL,
  jsonp = TRUE,
  as_plotly = FALSE,
  plotly_args = NULL,
  plotly_cfg = NULL,
  split_sig = NULL,
  self_contained = FALSE,
  thumb = TRUE,
facet_trelliscope

```r
auto_cog = FALSE,
split_layout = FALSE,
data = ggplot2::waiver()
)
```

**Arguments**

- `facets` formula to facet the panels on. Similar to `ggplot2::facet_wrap`'s `facets`
- `nrow` the number of rows of panels to display by default
- `ncol` the number of columns of panels to display by default
- `scales` should scales be the same ("same", the default), free ("free"), or sliced ("sliced"). May provide a single string or two strings, one for the X and Y axis respectively.
- `name` name of the display
- `group` group that the display belongs to
- `desc` description of the display
- `md_desc` optional string of markdown that will be shown in the viewer for additional context about the display
- `path` the base directory of the trelliscope application
- `height` height in pixels of each panel
- `width` width in pixels of each panel
- `state` the initial state the display will open in
- `jsonp` should json for display object be jsonp (TRUE) or json (FALSE)?
- `as_plotly` should the panels be written as plotly objects?
- `plotly_args` optional named list of arguments to send to `ggplotly`
- `plotly_cfg` optional named list of arguments to send to plotly's `config` method
- `split_sig` optional string that specifies the "signature" of the data splitting. If not specified, this is calculated as the md5 hash of the sorted unique facet variables. This is used to identify "related displays" - different displays that are based on the same faceting scheme. This parameter should only be specified manually if a display’s faceting is mostly similar to another display’s.
- `self_contained` should the Trelliscope display be a self-contained html document? (see note)
- `thumb` should a thumbnail be created?
- `auto_cog` should auto cogs be computed (if possible)?
- `split_layout` boolean that determines if the layout is split into components like a `facet_grid` vs. individual panels like `facet_wrap`. Only applies to `ggplot2` plot objects.
- `data` data used for faceting. Defaults to the first layer data

**Note**

Note that `self_contained` is severely limiting and should only be used in cases where you would either like your display to show up in the RStudio viewer pane, in an interactive R Markdown Notebook, or in a self-contained R Markdown html document.
Examples

```r
## Not run:
library(ggplot2)

# basically swap out facet_wrap for facet_trelliscope
qplot(cty, hwy, data = mpg) +
  facet_trelliscope(~ class + manufacturer)

# not required, but if you set labels, these will be added as
# descriptions to the cognostics that are automatically computed
mpg <- set_labels(mpg, mpg_labels)
qplot(cty, hwy, data = mpg) +
  theme_bw() +
  facet_trelliscope(~ manufacturer + class, nrow = 2, ncol = 4)

# using plotly
library(plotly)
qplot(cty, hwy, data = mpg) +
  theme_bw() +
  facet_trelliscope(~ manufacturer + class, nrow = 2, ncol = 4, as_plotly = TRUE)

qplot(class, cty, data = mpg, geom = c("boxplot", "jitter"), na.rm = TRUE) +
  facet_trelliscope(~ class, ncol = 7, height = 800, width = 200,
                   state = list(sort = list(sort_spec("cty_mean")))) +
  theme_bw()

library(gapminder)
qplot(year, lifeExp, data = gapminder) +
  xlim(1948, 2011) + ylim(10, 95) + theme_bw() +
  facet_trelliscope(~ country + continent, nrow = 2, ncol = 7,
                   width = 300, as_plotly = TRUE,
                   plotly_cfg = list(displayModeBar = FALSE))
## End(Not run)
```

---

### img_panel

**Cast a vector of URLs pointing to images as an image panel source**

**Description**

Cast a vector of URLs pointing to images as an image panel source

**Usage**

```r
img_panel(x)
```

**Arguments**

- `x` a vector of URLs pointing to images
img_panel_local

Cast a vector of URLs pointing to local images as an image panel source

Description

Cast a vector of URLs pointing to local images as an image panel source

Usage

img_panel_local(x)

Arguments

x

a vector of URLs pointing to images

Note

x must be paths relative to the path argument passed to trelliscope.

Examples

## Not run:
# assuming images are available locally in relative path pokemon_local/images
pokemon$img <- img_panel_local(paste0("images/", basename(pokemon$url_image)))
trelliscope(pokemon, name = "pokemon", path = "pokemon_local")
## End(Not run)

map2_cog

Map over multiple inputs simultaneously and return a vector of cognostics data frames

Description

Map over multiple inputs simultaneously and return a vector of cognostics data frames

Usage

map2_cog(.x, .y, .f, ...)

pmap_cog(.l, .f, ...)

Arguments

.x, .y  Vectors of the same length. A vector of length 1 will be recycled.
.f     A function, formula, or atomic vector (see map2 for details)
...    additional arguments passed on to .f.
.l     A list of lists. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.

Details

See map2

Examples

library(tidyr)
library(purrr)
library(plotly)
library(dplyr)

iris %>%
nest(data = ~Species) %>%
mutate(
  mod = map(data, ~ lm(Sepal.Length ~ Sepal.Width, data = .x)),
  cogs = map2_cog(data, mod, function(data, mod) {
    tibble(max_sl = max(data$Sepal.Length), slope = coef(mod)[2])
  }),
  panel = map2_plot(data, mod, function(data, mod) {
    plot_ly(data = data, x = ~Sepal.Width, y = ~Sepal.Length,
      type = "scatter", mode = "markers", name = "data") %>%
    add_trace(data = data, x = ~Sepal.Width, y = ~predict(mod),
      mode = "lines", name = "lm")
  })
) %>%
trelliscope(name = "iris")

map2_plot

Map over multiple inputs simultaneously and return a vector of plots

Description

Map over multiple inputs simultaneously and return a vector of plots

Usage

map2_plot(.x, .y)
map_cog

Arguments

- `.x, .y` Vectors of the same length. A vector of length 1 will be recycled.
- `.f` A function, formula, or atomic vector (see `map2` for details)
- `...` additional arguments passed on to `.f`.
- `.l` A list of lists. The length of `.l` determines the number of arguments that `.f` will be called with. List names will be used if present.

Details

See `map2`

Examples

```r
library(tidyr)
library(purrr)
library(plotly)
library(dplyr)

iris %>%
  nest(data = -Species) %>%
  mutate(
    mod = map(data, ~ lm(Sepal.Length ~ Sepal.Width, data = .x)),
    panel = map2_plot(data, mod, function(data, mod) {
      plot_ly(data = data, x = ~Sepal.Width, y = ~Sepal.Length,
        type = "scatter", mode = "markers", name = "data") %>%
      add_trace(data = data, x = ~Sepal.Width, y = ~predict(mod),
        mode = "lines", name = "lm")
    })) %>%
  trelliscope(name = "iris")
```

map_cog

Apply a function to each element of a vector and return a vector of cognostics data frames

Description

Apply a function to each element of a vector and return a vector of cognostics data frames

Usage

`map_cog(.x, .f, ...)`

Arguments

- `.x` a list or atomic vector (see `map` for details)
- `.f` a function, formula, or atomic vector (see `map` for details)
- `...` additional arguments passed on to `.f` (see `map` for details)
library(dplyr)
library(tidyr)
library(plotly)

`ggplot2::mpg` %>%
`nest(data = !one_of(c("manufacturer", "class")))` %>%
`mutate(`
  `cog = map_cog(data, function(x) tibble(mean_hwy = mean(x$hwy))),`
  `panel = map_plot(data, function(x) {
    `plot_ly(data = x, x = ~cty, y = ~hwy,`
    `type = "scatter", mode = "markers")`
  })`
)` %>%
`trelisscope(name = "city_vs_highway_mpg")

---

### `map_plot`

Apply a function to each element of a vector and return a vector of plots

#### Description

Apply a function to each element of a vector and return a vector of plots

#### Usage

`map_plot(.x, .f, ...)`

#### Arguments

- `.x`: a list or atomic vector (see `map` for details)
- `.f`: a function, formula, or atomic vector (see `map` for details)
- `...`: additional arguments passed on to `.f` (see `map` for details)

#### Details

See `map`
library(dplyr)
library(tidyr)
library(purrr)
library(plotly)
library(gapminder)

# nest gapminder data by country
by_country <- gapminder %>%
  nest(data = !one_of(c("country", "continent")))

# add in a plot column with map_plot
by_country <- by_country %>% mutate(
  panel = map_plot(data, function(x) {
    plot_ly(data = x, x = ~year, y = ~lifeExp,
      type = "scatter", mode = "markers") %>%
      layout(
        xaxis = list(range = c(1948, 2011)),
        yaxis = list(range = c(10, 95)))
  }))

# plot it
by_country %>%
  trelliscope("gapminder", nrow = 2, ncol = 7, width = 300)

# example using mpg data
ggplot2::mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(panel = map_plot(data, function(x) {
    plot_ly(data = x, x = ~hwy, y = ~cty,
      type = "scatter", mode = "markers")
  })) %>%
  trelliscope(name = "city_vs_highway_mpg")

mpg_labels

Labels for ggplot2 "mpg" data

Description

Labels for ggplot2 "mpg" data

Usage

mpg_labels

Format

An object of class list of length 10.
prepare_display

---

**panels**

*Panels Wrapper Function*

**Description**

Panels Wrapper Function

**Usage**

```r
panels(.x, .f, ...)
```

**Arguments**

- `.x` a list or atomic vector (see `map` for details)
- `.f` a function, formula, or atomic vector (see `map` for details)
- `...` additional arguments passed on to `.f` (see `map` for details)

**Details**

See `map`

**Examples**

```r
lplyr)
library(tidy)
library(plotly)
ggplot2::mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(panel = map_plot(data, function(x) {
    plot_ly(data = x, x = ~hwy, y = ~cty,
        type = "scatter", mode = "markers")
  }) %>%
  trelliscope(name = "city_vs_highway_mpg")
```

---

**prepare_display**

*Set up all auxiliary files needed for a Trelliscope app*

**Description**

Set up all auxiliary files needed for a Trelliscope app

**Usage**

```r
prepare_display(base_path, id, self_contained = FALSE, jsonp = TRUE, pb = NULL)
```
print.facet_trelliscope

Arguments

- **base_path**: the base directory of the trelliscope application
- **id**: a unique id for the application
- **self_contained**: should the Trelliscope display be a self-contained html document?
- **jsonp**: should json for display list and app config be jsonp (TRUE) or json (FALSE)?
- **pb**: optional progress bar object to pass in and use to report progress

Print facet trelliscope object

Description

Print facet trelliscope object

Usage

```r
# S3 method for class 'facet_trelliscope'
print(x, ...)
```

Arguments

- **x**: plot object
- **...**: ignored

set_labels

Set labels for a data frame

Description

Set labels for a data frame

Usage

```r
set_labels(dat, label_list)
```

Arguments

- **dat**: a data frame to apply labels to
- **label_list**: a named list with names matching those of dat and values being labels

Value

data frame with labels attached as attributes (attached to each column and named "label")
sort_spec  

Specify how a display should be sorted

Description

Specify how a display should be sorted

Usage

sort_spec(name, dir = "asc")

Arguments

name  variable name to sort on
dir  direction to sort ('asc' or 'desc')

trelliscope  

Create a Trelliscope Display

Description

Create a Trelliscope Display

Usage

trelliscope(
  x,
  name,
  group = "common",
  panel_col = NULL,
  desc = "",
  md_desc = "",
  path,
  height = 500,
  width = 500,
  auto_cog = FALSE,
  state = NULL,
  nrow = 1,
  ncol = 1,
  jsonp = TRUE,
  split_sig = NULL,
  self_contained = FALSE,
  thumb = FALSE
)
Arguments

- **x**: an object to create at `trelliscope` display for
- **name**: name of the display
- **group**: group that the display belongs to
- **panel_col**: optional string specifying the column to use for panels (if there are multiple plot columns in `x`)
- **desc**: optional text description of the display
- **md_desc**: optional string of markdown that will be shown in the viewer for additional context about the display
- **path**: the base directory of the `trelliscope` application
- **height**: height in pixels of each panel
- **width**: width in pixels of each panel
- **auto_cog**: should auto cogs be computed (if possible)?
- **state**: the initial state the display will open in
- **nrow**: the number of rows of panels to display by default
- **ncol**: the number of columns of panels to display by default
- **jsonp**: should json for display object be jsonp (TRUE) or json (FALSE)?
- **split_sig**: optional string that specifies the "signature" of the data splitting. If not specified, this is calculated as the md5 hash of the sorted unique facet variables. This is used to identify "related displays" - different displays that are based on the same faceting scheme. This parameter should only be specified manually if a display's faceting is mostly similar to another display's.
- **self_contained**: should the `Trelliscope` display be a self-contained html document? (see note)
- **thumb**: should a thumbnail be created?

Note

Note that `self_contained` is severely limiting and should only be used in cases where you would either like your display to show up in the RStudio viewer pane, in an interactive R Markdown Notebook, or in a self-contained R Markdown html document.

Examples

```r
## Not run:
library(dplyr)
library(tidyr)
library(purrr)
library(plotly)
library(ggplot2)

# tidyverse + plotly
d <- mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(
```
```r
mean_city mpg = map_dbl(data, ~ mean(.$cty)),
panel = map_plot(data, function(x) {
  plot_ly(data = x, x = ~cty, y = ~hwy,
  type = "scatter", mode = "markers")
})
}

d %>% trelliscope(name = "city_vs_highway_mpg")

# set default layout
d %>% trelliscope(name = "city_vs_highway_mpg", nrow = 2, ncol = 3)

# set the output path for where files will be stored
my_displays <- tempfile()
d %>% trelliscope(name = "city_vs_highway_mpg", path = my_displays)

# multiple displays can be added to the same path and all will be available in the viewer
d %>% trelliscope(name = "city_vs_highway_mpg2", path = my_displays)

# ordering the data frame will set default sort order of the display
d %>%
  arrange(-mean_city_mpg) %>%
  trelliscope(name = "city_vs_highway_mpg")

# tidyverse + ggplot2
mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(
    panel = map_plot(data, ~
      qplot(cty, hwy, data = ., xlab("cty") + ylab("hwy") +
      xlim(7, 37) + ylim(9, 47) + theme_bw())
  ) %>%
  trelliscope(name = "tidy_gg")

# computing additional cognostics
mpg_cog %<-
  mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(
    cogs = map_cog(data, ~
      tibble(
        mean_city_mpg = mean(.$cty),
        mean_hwy_mpg = mean(.$hwy),
        most_common_drv = tail(names(table(.$drv)), 1)
      ))
  )

# computing additional cognostics explicitly using cog()
# so we can specify descriptions, etc.
mpg_cog2 %<-
  mpg %>%
  nest(data = !one_of(c("manufacturer", "class"))) %>%
  mutate(
    cogs = map_cog(data, ~
      tibble(
        mean_city_mpg = cog(mean(.$cty), desc = "Mean city mpg"),
        mean_hwy_mpg = cog(mean(.$hwy), desc = "Mean highway mpg"),
        most_common_drv = cog(tail(names(table(.$drv)), 1), desc = "Most common drive type")
      ))
  )
```
```r
panel = map_plot(data, function(x) {
  plot_ly(data = x, x = ~cty, y = ~hwy,
         type = "scatter", mode = "markers") %>%
  layout(
    xaxis = list(range = c(9, 47)),
    yaxis = list(range = c(7, 37))
  )
})

mpg_cog2 %>%
  trelliscope(name = "city_vs_highway_mpg", nrow = 1, ncol = 2)

## End(Not run)
```

---

**Trelliscope-shiny**  
*Shiny bindings for Trelliscope*

**Description**

Output and render functions for using trelliscopejs_widget within Shiny applications and interactive Rmd documents.

**Usage**

```r
trelliscopeOutput(outputId, width = "100\%", height = "400px")

renderTrelliscope(expr, env = parent.frame(), quoted = FALSE)
```

**Arguments**

- `outputId`  
  output variable to read from

- `width, height`  
  Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.

- `expr`  
  An expression that generates a trelliscopejs_widget

- `env`  
  The environment in which to evaluate expr.

- `quoted`  
  Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.
**update_display_list**  
*Update Trelliscope app display list file*

**Description**

Update Trelliscope app display list file

**Usage**

```r
update_display_list(base_path, jsonp = TRUE)
```

**Arguments**

- `base_path`: the base directory of the trelliscope application
- `jsonp`: should json for display list be jsonp (TRUE) or json (FALSE)?

---

**write_cognostics**  
*Write cognostics data for a display in a Trelliscope app*

**Description**

Write cognostics data for a display in a Trelliscope app

**Usage**

```r
write_cognostics(cogdf, base_path, id, name, group = "common", jsonp = TRUE)
```

**Arguments**

- `cogdf`: a data frame of cognostics, prepared with `as_cognostics`
- `base_path`: the base directory of the trelliscope application
- `id`: a unique id for the application
- `name`: name of the display
- `group`: group that the display belongs to
- `jsonp`: should json for cognostics be jsonp (TRUE) or json (FALSE)?
write_config

**Write Trelliscope app configuration file**

**Description**

Write Trelliscope app configuration file

**Usage**

```r
write_config(
  base_path,
  id,
  self_contained = FALSE,
 jsonp = TRUE,
  split_layout = FALSE,
  has_legend = FALSE
)
```

**Arguments**

- `base_path`: the base directory of the trelliscope application
- `id`: a unique id for the application
- `self_contained`: should the Trelliscope display be a self-contained html document?
- `jsonp`: should json for app config be jsonp (TRUE) or json (FALSE)?
- `split_layout`: boolean that determines if the layout is split into components like a facet_grid vs. individual panels like facet_wrap. Only applies to ggplot2 plot objects.
- `has_legend`: should a legend be reported for split_layout

write_display_obj

**Write a "display object" file for a Trelliscope app**

**Description**

Write a "display object" file for a Trelliscope app

**Usage**

```r
write_display_obj(
  cogdf,
  panel_example,
  base_path,
  id,
  name,
  group = "common",
)```
desc = "", height = 500, width = 500, md_desc = "", state = NULL, jsonp = TRUE, split_sig = NULL, panel_img_col = NULL, selfcontained = FALSE, thumb = TRUE, split_layout = FALSE, split_aspect = NULL, has_legend = FALSE, pb = NULL)

Arguments

cogdf a data frame of cognostics, prepared with as_cognostics
panel_example an example object of one panel of a display (can be trellis, ggplot2, or htmlwidget object)
base_path the base directory of the trelliscope application
id a unique id for the application
name name of the display
group group that the display belongs to
desc description of the display
height height in pixels of each panel
width width in pixels of each panel
md_desc optional string of markdown that will be shown in the viewer for additional context about the display
state the initial state the display will open in
jsonp should json for display object be jsonp (TRUE) or json (FALSE)?
split_sig optional string "signature" specifying the data splitting
panel_img_col which column (if any) is a panel image column?
self_contained should the Trelliscope display be a self-contained html document?
thumb should a thumbnail be created?
split_layout boolean that determines if the layout is split into components like a facet_grid vs. individual panels like facet_wrap. Only applies to ggplot2 plot objects.
split_aspect list indicating aspect ratios of axes for a split layout. Only applies to ggplot2 plot objects.
has_legend should a legend be reported for split_layout
pb optional progress bar object to pass in and use to report progress
write_panel  

Write a plot object as a panel in a Trelliscope display

Description

Write a plot object as a panel in a Trelliscope display

Usage

write_panel(
    plot_object,
    key,
    base_path,
    name,
    group = "common",
    width,
    height,
    jsonp = TRUE,
    split_layout = FALSE
)

Arguments

plot_object  a plot object to be written (can be trellis, ggplot2, or htmlwidget)
key  a string identifying the panel key, which will be used as the panel file name and which the panelKey column of the cognostics data frame should point to
base_path  the base directory of the trelliscope application
name  name of the display that the panel belongs to
group  group name of the display that the panel belongs to
width  width in pixels of each panel
height  height in pixels of each panel
jsonp  should json for panel be jsonp (TRUE) or json (FALSE)?
split_layout  boolean that determines if the layout is split into components like a facet_grid vs. individual panels like facet_wrap. Only applies to ggplot2 plot objects.
write_panels  Write a list of plot objects as panels in a Trelliscope display

Description
Write a list of plot objects as panels in a Trelliscope display

Usage
write_panels(plot_list, ..., pb = NULL)

Arguments
plot_list  a named list of plot objects to be written as panels (objects can be trellis, ggplot2, or htmlwidget) with the list names being the keys for the panels
...  params passed directly to write_panel
pb  optional progress bar object to pass in and use to report progress
Index

* datasets
  mpg_labels, 15
  as_cognostics, 3, 22, 24
  cog, 3, 7
  cog_disp_filter, 6
  cog_href, 7
  cogs, 5

  facet_trelliscope, 8
  facet_wrap, 9

  img_panel, 10
  img_panel_local, 11

  map, 5, 13, 14, 16
  map2, 12, 13
  map2_cog, 11
  map2_plot, 12
  map_cog, 13
  map_plot, 14
  mpg_labels, 15

  panels, 16
  pmap_cog (map2_cog), 11
  pmap_plot (map2_plot), 12
  prepare_display, 16
  printfacet_trelliscope, 17

  renderTrelliscope (Trelliscope-shiny), 21

  set_labels, 17
  sort_spec, 18

  trelliscope, 11, 18
  Trelliscope-shiny, 21
  trelliscopejs-package, 2
  trelliscopeOutput (Trelliscope-shiny), 21

update_display_list, 22
write_cognostics, 22
write_config, 23
write_display_obj, 23
write_panel, 25, 26
write_panels, 26

27