Package 'tidycharts'

August 23, 2021

Type  Package
Title  Generate Tidy Charts Inspired by 'IBCS'
Version 0.1.2
Maintainer Bartosz Sawicki <sawicki.bartosz@interia.pl>

Description  There is a wide range of R packages created for data visualization, but still, there was no simple and easily accessible way to create clean and transparent charts - up to now. The 'tidycharts' package enables the user to generate charts compliant with International Business Communication Standards ('IBCS').

It means unified bar widths, colors, chart sizes, etc. Creating homogeneous reports has never been that easy! Additionally, users can apply semantic notation to indicate different data scenarios (plan, budget, forecast). What's more, it is possible to customize the charts by creating a personal color pallet with the possibility of switching to default options after the experiments.

We wanted the package to be helpful in writing reports, so we also made joining charts in a one, clear image possible.

All charts are generated in SVG format and can be shown in the 'RStudio' viewer pane or exported to HTML output of 'knitr'/markdown'.

License  GPL (>= 3)
Encoding  UTF-8
RoxygenNote  7.1.1

Imports  magick, rsvg, rlang, testthat, dplyr, methods, graphics, htmlwidgets, lubridate, stringr

Depends  magrittr,
Suggests  knitr, rmarkdown, palmerpenguins, tidyverse, covr
VignetteBuilder  knitr

NeedsCompilation  no

Author  Przemysław Biecek [aut] (<https://orcid.org/0000-0001-8423-1823>),
Piotr Piątyszek [aut],
Kinga Ułasik [aut],
Bartosz Sawicki [aut, cre]

Repository  CRAN
Date/Publication  2021-08-23 10:20:02 UTC
R topics documented:

add_bars ....................................................... 3
add_title .......................................................... 4
add_waterfall_bars ............................................. 5
bar_chart ............................................................. 6
bar_chart_absolute_variance .................................. 7
bar_chart_grouped ............................................... 8
bar_chart_normalized .......................................... 9
bar_chart_reference ........................................... 10
bar_chart_relative_variance .................................. 11
bar_chart_waterfall ........................................... 12
column_chart ...................................................... 13
column_chart_absolute_variance ................................ 14
column_chart_grouped .......................................... 15
column_chart_normalized ........................................ 16
column_chart_reference ......................................... 17
column_chart_relative_variance ................................ 19
column_chart_waterfall ......................................... 20
column_chart_waterfall_variance ............................... 21
draw_triangle ....................................................... 22
facet_chart .......................................................... 22
get_gray_color_stacked ........................................... 23
get_vector ............................................................ 24
join_charts .......................................................... 24
line_chart ............................................................ 25
line_chart_dense .................................................... 27
line_chart_dense_custom ......................................... 28
line_chart_markers ............................................. 29
line_chart_markers_reference ................................ 31
line_chart_normalized ........................................... 32
line_chart_stacked ............................................... 33
parse_time_series ............................................. 34
restore_defaults ................................................... 35
scatter_plot ......................................................... 36
set_colors .......................................................... 37
set_styles .......................................................... 38
str_width ............................................................ 39
SVGRenderer ........................................................ 39
SVGsave ............................................................. 40

Index 41
Description

add bars to svg string

Usage

add_bars(
    svg_string,  
    df,  
    x,  
    series,  
    bar_width,  
    styles = NULL,  
    x_offset = 0,  
    translate = c(0, 0),  
    add_x_axis = TRUE,  
    color = NULL,  
    add_legend = FALSE,  
    legend_position = "left_top",  
    max_val = NULL
)

Arguments

- svg_string: the svg string to be appended, need to be finalized after
- df: data to be plotted - data frame in wide format
- x: vector to be on x axis
- series: character vector of column names representing series to split bars by it
- bar_width: the width of plotted bar
- styles: vector of styles of the bars
- x_offset: how much bars should be offset to the right (negative value means offsetting to the left)
- translate: vector of translation of the bars from the origin
- add_x_axis: boolean flag, if true automatically adds x axis with label
- color: optional custom color of the bars series, in svg string format, i.e.: "rgb(223,12,121)" or "black"
- add_legend: boolean flag if legend should be added
- legend_position: string with legend position
- max_val: maximal value that bars will be scaled to
add_title

Value

svg string with added bars

Description

Add IBCS compliant legend

Usage

add_title(svg_string, line1, line2_measure, line2_rest, line3 = "")

Arguments

svg_string one element character vector containing SVG graphic statements. Legend will be added to this plot.
line1 first line of title. Element(s) of the structure dimension represent the object of the report, typically a legal entity, an organization unit, or a line of business
line2_measure First part of second line of the title. It will be in bold text. It should represent business measure being analyzed.
line2_rest Second part of second line of the title. It should represent units of measure.
line3 Third line of the title, it should indicate time, scenarios, variances, etc

Value

one element character vector containing SVG graphic statements. svg_string with appended legend.

Examples

df <- data.frame(x = 2010:2015, sales = rnorm(6, 10, 2))
column_chart(df, df$x, 'sales') %>%
  add_title(line1 = 'Department of Big Computers',
            line2_measure = "Sales",
            line2_rest = "in mEUR",
            line3 = "2010..2015") %>%
  SVGRenderer()
add_waterfall_bars

Add waterfall style bars to the column chart

Description

Add waterfall style bars to the column chart

Usage

```r
add_waterfall_bars(
svg_string,
df,
x,
series,
bar_width,
styles = NULL,
pos_color = "rgb(64,64,64)",
neg_color = "black",
add_result_bar = TRUE,
result_bar_pos = "1",
positive_prefix = "",
result_bar_color = NULL,
result_title = NULL,
ref_value = 0,
translate_vec = c(0, 0)
)
```

Arguments

- `svg_string` the svg string to be appended, need to be finalized after
- `df` data to be plotted - data frame in wide format
- `x` vector to be on x axis
- `series` character vector of column names representing series to split bars by it
- `bar_width` the width of plotted bar
- `styles` vector of styles of the bars
- `pos_color` color to be associated with positive values (in string format)
- `neg_color` color to be associated with negative values (in string format)
- `add_result_bar` boolean flag to add result bar as the last bar or not.
- `result_bar_pos` flag indicating position of the result bar. 1 - bar offset 1/9 category width right from the last bar. 2 - result bar as completely new bar. If add_result_bar is false, it is ignored.
- `positive_prefix` how to indicate positive value, i.e. "+" or "" (empty string)."
result_bar_color  
   color of result bar. If add_result_bar is false, it is ignored.
result_title   
   title of result bar to be on x axis. If add_result_bar is false, it is ignored.
ref_value      
   first bar starts from this value, intended to be used with add_first_bar function.
translate_vec  
   2 element translation vector. By setting this parameter you can translate bars and legend.

Value

svg string with appended waterfall bars

---

**bar_chart**

Generates basic horizontal barchart. If more than one series is supplied, stacked barchart is generated.

Description

Generates basic horizontal barchart. If more than one series is supplied, stacked barchart is generated.

Usage

```
bar_chart(data, cat, series, series_labels = series, styles = NULL)
```

Arguments

- **data**
  - data frame containing data to be plotted
- **cat**
  - vector containing category names of values
- **series**
  - vector containing names of columns in data with values to plot
- **series_labels**
  - vector containing names of series to be shown on the plot
- **styles**
  - optional vector with styles of bars

Value

SVG string containing chart

Examples

```r
#prepare the data frame
data <- data.frame(
Products = c(538, 250, 75, 301, 227, 100, 40),
Services = c(621, 545, 302, 44, 39, 20, 34)
)
#generate svgstring
barchart <- bar_chart(data, data$city, c("Products", "Services"), c("Products", "Services"))
```
Generate bar chart with absolute variance

Description

Visualize variance between baseline and real in absolute units. Choose colors parameter accordingly to business interpretation of larger/smaller values.

Usage

```r
bar_chart_absolute_variance(
  cat,
  baseline,
  real,
  colors = 1,
  data = NULL,
  y_title,
  y_style = "previous"
)
```

Arguments

- `cat`: vector containing category names of values
- `baseline`: vector containing base values or name of column in data with base values
- `real`: vector containing values that will be compared to baseline or name of column in data with that values
- `colors`: 1 if green color represents positive values having good business impact and red negative values having bad impact or 2 if otherwise
- `data`: data frame with columns containing data for x, baseline or real series
- `y_title`: title of the series values
- `y_style`: style of y axis to indicate baseline scenario

Value

SVG string containing chart
Examples

```r
# get some data
real <- sin(1:5)
baseline <- cos(1:5)
cat <- letters[1:5]

bar_chart_absolute_variance(
  cat = cat,
  baseline = baseline,
  real = real,
  y_title = 'a title'
) %>%
SVGrenderer() # show the plot
```

---

`bar_chart_grouped` *Generates grouped horizontal bar chart with scenario triangles.*

Description

Generates grouped horizontal bar chart with scenario triangles.

Usage

```r
bar_chart_grouped(
  data, 
  cat, 
  foreground, 
  background, 
  markers = NULL, 
  series_labels, 
  styles = NULL
)
```

Arguments

- `data` data frame in wide format containing data to be plotted
- `cat` vector containing category names of values
- `foreground` vector or name of column in data representing heights of bars visible in the foreground
- `background` vector or name of column in data representing heights of bars visible in the background
- `markers` optional vector representing position of triangles
- `series_labels` vector of series titles. Consists of 2 or 3 elements
- `styles` optional data frame of styles. First column contains styles for foreground series, second for background, third for triangles. `dim(styles)` must be `length(x)`, `length(titles)`
**Value**

SVG string containing chart

**Examples**

```r
# preparing data frames
data <- data.frame(
  city = c("Berlin", "Paris", "London", "Munich", "Vienna"),
  AC = c(592, 1166, 618, 795, 538),
  PL = c(570, 950, 800, 780, 460),
  triangles = c(545, 800, 900, 600, 538) # AC toten bardziej na wierzchu
)

data <- data.frame(
  AC = c("actual","actual","actual","actual","actual"),
  PL = c("plan","plan","plan","plan","plan"),
  triangles = c("previous","previous","previous","previous","previous"))

# creating the svg string
barchart_grouped <- bar_chart_grouped(data,
  data$city, "AC", "PL", "triangles", c("triangles", "AC", "PL"), df_styles)

# showing the plot
barchart_grouped %>% SVGrenderer()
```

---

**bar_chart_normalized**

**Description**

Generates normalized horizontal barchart. If more than one series is supplied, stacked barchart is generated.

**Usage**

```r
bar_chart_normalized(data, cat, series, series_labels = series)
```

**Arguments**

- `data`: data frame containing data to be plotted
- `cat`: vector containing category names of values
- `series`: vector containing names of columns in data with values to plot
- `series_labels`: vector containing names of series to be shown on the plot
Examples

```r
# prepare the data frame
data <- data.frame(
  Products = c(538, 250, 75, 301, 227, 100, 40),
  Services = c(621, 545, 302, 44, 39, 20, 34)
)
# create svg string
barchart_normalized <- bar_chart_normalized(
  data = data,
  cat = data$city,
  series = c("Products", "Services")
)
# show the plot
barchart_normalized %>% SVGrenderer()
```

---

**Description**

Generates basic horizontal barchart with index on a given value. If more than one series is supplied, stacked barchart is generated.

**Usage**

```r
bar_chart_reference(
  data, 
  cat, 
  series, 
  ref_val, 
  series_labels = series, 
  styles = NULL, 
  ref_label = ref_val
)
```

**Arguments**

- **data**: data frame containing data to be plotted
- **cat**: vector containing category names of values
- **series**: vector containing names of columns in data with values to plot
bar_chart_relative_variance

- **ref_val**: numeric value of the index
- **series_labels**: vector containing names of series to be shown on the plot
- **styles**: optional vector with styles of bars
- **ref_label**: string defining a text that should be displayed in the referencing line. Set by default to index_val.

**Value**

SVG string containing chart

**Examples**

```r
# prepare the data frame
data <- data.frame(
  Products = c(538, 250, 75, 301, 227, 100, 40),
  Services = c(621, 545, 302, 44, 39, 20, 34)
)
# create svg string
barchart_ref <- bar_chart_reference(data, data$city, c("Products"), 100, c("Products"))

# show the plot
barchart_ref %>% SVGrenderer()
```

---

**Description**

Generate bar chart with relative variance (in percents)

**Usage**

```r
bar_chart_relative_variance(
  cat,
  baseline,
  real,
  colors = 1,
  data = NULL,
  y_title,
  y_style = "previous",
  styles = NULL
)
```
Arguments

- **cat**: vector containing category names of values
- **baseline**: vector containing base values or name of column in data with base values
- **real**: vector containing values that will be compared to baseline or name of column in data with that values
- **colors**: 1 if green color represents positive values having good business impact and red negative values having bad impact or 2 if otherwise
- **data**: data frame with columns containing data for x, baseline or real series
- **y_title**: title of the series values
- **y_style**: style of y axis to indicate baseline scenario
- **styles**: optional vector with styles of the pin heads

Value

SVG string containing chart

Examples

```r
# get some data
defined <- sin(1:5)
baseline <- cos(1:5)
cat <- letters[1:5]

bar_chart_relative_variance(
  cat = cat,
  baseline = baseline,
  real = real,
  y_title = 'a title'
) %>%
  SVGrenderer() # show the plot
```

---

**bar_chart_waterfall**  
*Generate horizontal waterfall chart*

Description

Generate horizontal waterfall chart

Usage

```r
bar_chart_waterfall(
  cat,
  series,
  data = NULL,
  add_result = FALSE,
  result_title = NULL
)
```
**Arguments**

- **cat**: vector containing category names of values
- **series**: vector containing names of columns in data with values to plot
- **data**: data frame containing data to be plotted
- **add_result**: boolean value if result bar should be plotted
- **result_title**: the title for the result bar. Ignored if add_result is false

**Value**

SVG string containing chart

**Examples**

```r
df <- data.frame(
  profit = sin(1:7)
)

bar_chart_waterfall(cat = 'city', series = 'profit', data = df) %>% SVGrenderer()
```

---

**Description**

Generate basic column chart. If more than one series is supplied, stacked column plot is generated

**Usage**

```r
column_chart(
  data, x,
  series = NULL,
  series_labels = series,
  styles = NULL,
  interval = "months"
)
```

**Arguments**

- **data**: data frame in wide format containing data to be plotted
- **x**: vector containing labels for x axis or name of column in data with values of x axis labels
- **series**: vector containing names of columns in data with values to plot
**series_labels**  optional vector with labels for series to be plotted as legend. The default is the same as series.

**styles**  optional vector with styles of bars

**interval**  intervals on x axis. The width of the bars depends on this parameter

**Value**

SVG string containing chart

**Examples**

```r
# prepare some data frame
df <- data.frame(x = month.abb[1:6],
                 y = c(2, 4, 2, 1, 2.5, 3),
                 z = c(3, 4.5, 2, 1, 4, 2))

# generate character vectors with svg data
svg1 <- column_chart(df, x = 'x', series = 'y')
svg2 <- column_chart(df, x = df$x, series = c('y', 'z'))

# show the plot
svg1 %>% SVGrenderer()
```

---

**Description**

Visualize variance between two time series (baseline and real) in the same units as the time series. Choose colors parameter accordingly to business interpretation of larger/smaller values.

**Usage**

```r
column_chart_absolute_variance(
  x, baseline, real, colors = 1, data = NULL,
  x_title = "PY", x_style = "previous",
  interval = "months"
)
```
Arguments

- **x**: vector containing labels for x axis or name of column in data with values of x axis labels
- **baseline**: vector containing base values or name of column in data with base values
- **real**: vector containing values that will be compared to baseline or name of column in data with that values
- **colors**: 1 if green color represents positive values having good business impact and red negative values having bad impact or 2 if otherwise
- **data**: data frame with columns containing data for x, baseline or real series
- **x_title**: the title of the plot
- **x_style**: style of the x axis to indicate baseline scenario. The default is 'previous'.
- **interval**: intervals on x axis. The width of the bars depends on this parameter

Value

SVG string containing chart

Examples

```r
x <- month.abb
baseline <- rnorm(12)
real <- c(rnorm(6, mean = -1), rnorm(6, mean = 1))
column_chart_absolute_variance(x, baseline, real, x_title = 'profit') %>%
  SVGrenderer()
```

Description

Generate grouped column chart for visualizing up to 3 data series

Usage

```r
column_chart_grouped(
  x,
  foreground,
  background,
  markers = NULL,
  data = NULL,
  series_labels,
  styles = NULL,
  interval = "months"
)
```
Arguments

- **x**: vector containing labels for x axis or name of column in data with values of x axis labels
- **foreground**: vector or name of column in data representing heights of bars visible in the foreground
- **background**: vector or name of column in data representing heights of bars visible in the background
- **markers**: optional vector representing position of triangles
- **data**: data frame in wide format containing data to be plotted
- **series_labels**: vector of series titles. Consists of 2 or 3 elements
- **styles**: optional dataframe of styles. First column contains styles for foreground series, second for background, third for triangles. \( \text{dim}(\text{styles}) \) must be \( \text{length}(x) \), \( \text{length}(\text{titles}) \)
- **interval**: intervals on x axis. The width of the bars depends on this parameter

Value

SVG string containing chart

Examples

```r
df <- data.frame(x = month.abb[7:12],
                 actual = rnorm(6, mean = 5, sd = 0.3),
                 budget = rnorm(6, mean = 4.5, sd = 0.7),
                 prev_year = rnorm(6, mean = 4))
column_chart_grouped(x = df$x, 
                      foreground = df$actual, 
                      background = df$budget, 
                      markers = df$prev_year, 
                      series_labels = c('AC', 'BU', 'PY')) %>% SVGrenderer()
```

---

**column_chart_normalized**

Generate column chart with normalization. Every column will be rescaled, so columns have the same height.

Description

Generate column chart with normalization. Every column will be rescaled, so columns have the same height.
Usage

column_chart_normalized(
  data,
  x,
  series = NULL,
  series_labels = series,
  interval = "months"
)

Arguments

data data frame in wide format containing data to be plotted
x vector containing labels for x axis or name of column in data with values of x
  axis labels
series vector containing names of columns in data with values to plot
series_labels optional vector with labels for series to be plotted as legend. The default is the
  same as series.
interval intervals on x axis. The width of the bars depends on this parameter

Value

SVG string containing chart

Examples

# prepare some data frame
df <- data.frame(x = month.abb[1:6],
  y = c(2, 4, 2, 1, 2.5, 3),
  z = c(3, 4.5, 2, 1, 4, 2))

# generate character vector with svg data
svg <- column_chart_normalized(df, x = df$x, series = c('y', 'z'))

# show the plot
svg %>% SVGrenderer()
Usage
column_chart_reference(
  data,
  x,
  series,
  ref_value,
  ref_label = NULL,
  styles = NULL,
  interval = "months"
)

Arguments
data data frame in wide format containing data to be plotted
x vector containing labels for x axis or name of column in data with values of x axis labels
series vector containing names of columns in data with values to plot
ref_value one element numeric vector with referencing value.
ref_label name of the referencing value
styles optional vector with styles of bars
interval intervals on x axis. The width of the bars depends on this parameter

Value
SVG string containing chart

Examples
# prepare some data frame
df <- data.frame(x = month.abb[1:6],
  y = c(2, 4, 2, 1, 2.5, 3),
  z = c(3, 4.5, 2, 1, 4, 2))

# generate character vector with svg data
svg <- column_chart_reference(df, x = 'x',
  series = 'y',
  ref_value = 3,
  ref_label = 'baseline')

# show the plot
svg %>% SVGrenderer()
column_chart_relative_variance

Generate column chart with relative variance (in percents)

Description

Generate column chart with relative variance (in percents)

Usage

column_chart_relative_variance(
  x,
  baseline,
  real,
  colors = 1,
  data = NULL,
  x_title,
  x_style = "previous",
  styles = NULL,
  interval = "months"
)

Arguments

x vector containing labels for x axis or name of column in data with values of x axis labels
baseline vector containing base values or name of column in data with base values
real vector containing values that will be compared to baseline or name of column in data with that values
colors 1 if green color represents positive values having good business impact and red negative values having bad impact or 2 if otherwise
data data frame with columns containing data for x, baseline or real series
x_title the title of the plot
x_style style of the x axis to indicate baseline scenario. The default is 'previous'.
styless optional vector with styles of the pin heads
interval intervals on x axis. The width of the bars depends on this parameter

Value

SVG string containing chart
Examples

```r
x <- month.abb
baseline <- rnorm(12, mean = 1, sd = 0.2)
real <- c(rnorm(6, mean = 0.8, sd = 0.2), rnorm(6, mean = 1.2, sd = 0.2))
column_chart_relative_variance(x, baseline, real, x_title = 'profit %') %>%
SVGrenderer()
```

---

column_chart_waterfall

Generate column waterfall chart for visualizing contribution.

Description

Generate column waterfall chart for visualizing contribution.

Usage

```r
column_chart_waterfall(data, x, series, styles = NULL, interval = "months")
```

Arguments

data data frame in wide format containing data to be plotted

x vector containing labels for x axis or name of column in data with values of x axis labels

series vector containing names of columns in data with values to plot

styles optional vector with styles of bars

interval intervals on x axis. The width of the bars depends on this parameter

Value

SVG string containing chart

Examples

```r
df <- data.frame(x = 10:18,
         y = rnorm(9))
column_chart_waterfall(df, 'x', 'y') %>% SVGrenderer()
```
column_chart_waterfall_variance

*Generate column waterfall chart with absolute variance*

**Description**

Generate column waterfall chart with absolute variance

**Usage**

```r
column_chart_waterfall_variance(
  x,
  baseline,
  real,
  colors = 1,
  data = NULL,
  result_title,
  interval = "months"
)
```

**Arguments**

- `x` vector containing labels for x axis or name of column in data with values of x axis labels
- `baseline` vector containing base values or name of column in data with base values
- `real` vector containing values that will be compared to baseline or name of column in data with that values
- `colors` 1 if green color represents positive values having good business impact and red negative values having bad impact or 2 if otherwise
- `data` data frame with columns containing data for x, baseline or real series
- `result_title` title for the result bar
- `interval` intervals on x axis. The width of the bars depends on this parameter

**Value**

SVG string containing chart

**Examples**

```r
x <- month.abb
baseline <- rnorm(12)
real <- c(rnorm(6, mean = -1), rnorm(6, mean = 1))
column_chart_waterfall_variance(x, baseline, real, result_title = 'year profit') %>% SVGrenderer()
```
**draw_triangle**

*Draw triangle and append it to svg string*

**Description**

Draw triangle and append it to svg string

**Usage**

```r
draw_triangle(
  svg_string,
  tip_position_x, tip_position_y,
  orientation = "left",
  style = NULL,
  translate_vec = c(0, 0)
)
```

**Arguments**

- `svg_string` : svg string to paste a trinagle
- `tip_position_x, tip_position_y` : x, y position of tip of the triangle
- `orientation` : where the triangle should be pointing. One of c('top', 'right', 'bottom', 'left').
- `style` : style of the triangle
- `translate_vec` : the translation vector

**Value**

svg string

---

**facet_chart**

*Facet chart*

**Description**

Create multiple charts with data split into groups

**Usage**

```r
facet_chart(data, facet_by, ncols = 3, FUN, ...)
```
get_gray_color_stacked

Arguments

- **data**: data frame in wide format containing data to be plotted
- **facet_by**: a name of column in data, that the charts will be splitted by
- **ncols**: number of columns of the plots. Number of rows will be adjusted accordingly
- **FUN**: function to plot the basic chart
- **...**: other parameters passed to FUN

Value

Character vector with SVG content

Examples

```r
facet_chart(
  data = mtcars,
  facet_by = 'cyl',
  ncols = 2,
  FUN = scatter_plot,
  x = mtcars$hp,
  y = mtcars$qsec,
  legend_title = ''
) %>% SVGrenderer()
```

get_gray_color_stacked

Function to get bar/area color for stacked plots.

Description

Function to get bar/area color for stacked plots.

Usage

```r
get_gray_color_stacked(series_number, colors_df = pkg.env$colors_df)
```

Arguments

- **series_number**: what is the number of the series. one of 1:6.
- **colors_df**: data frame with variety of colors

Value

list with bar_color and text_color
**get_vector**  
*Helper function to get the vector or column form df. If vector is passed it returns it. If name of column is passed, it returns the column as a vector.*

**Description**  
Helper function to get the vector or column form df. If vector is passed it returns it. If name of column is passed, it returns the column as a vector.

**Usage**  
```r
get_vector(df, vec)
```

**Arguments**
- `df`: data frame with a column
- `vec`: name of the column in df or vector of values

**Value**
- vector

**join_charts**  
*Join SVG charts This function first populates each place in the first row, then columns in the second row*

**Description**  
Join SVG charts This function first populates each place in the first row, then columns in the second row.

**Usage**
```r
join_charts(  
  ...,  
  nrows = max(length(list(...)), length(list_of_plots)),  
  ncols = 1,  
  list_of_plots = NULL
)
```
**line_chart**

Arguments

- ... multiple character vectors with SVG content
- nrows number of rows of plots in joint plot, default is set to number of plots
- ncols number of columns of plots in joint plot, default is set to 1
- list_of_plots optional list of plots to join. Use exclusively ... params or list_of_plots. Names of list entries will be plotted as titles of the plots

Value

Character vector with SVG content

Examples

```r
df <- data.frame(
  mon = month.abb[1:6],
  values = rnorm(6)
)

join_charts(
  column_chart(df, x = "mon", series = "values"),
  column_chart(df, x = "mon", series = "values")
) %>% SVGrenderer()
```

Description

Generates a line plot with markers on chosen points. Allows only one point per time interval. To create a plot with many points within one time interval try line_plot_with_many_points_complex().

Usage

```r
line_chart(
  data,
  x,
  series,
  series_labels,
  ser_names,
  point_cords,
  interval = "months"
)
```
Arguments

data  data frame containing data to be plotted
x     vector containing time intervals of the values
series vector containing names of columns in data with values to plot
series_labels vector containing names of series to be shown on the plot
ser_names vector containing column names of a value to be marked
point_cords vector of the same length as ser_names containing numerical values of indexes in data of values to be marked
interval intervals on x axis. The width of the bars depends on this parameter

Value

SVG string containing chart

Examples

#preparing the data frame
data <- data.frame(
  Gamma = c(98, 80, 16, 25, 55, 48),
  Delta = c(22, 25, 67, 73, 102, 98)
)

#defining rest of arguments
names <- c("Gamma", "Gamma", "Gamma", "Gamma", "Delta", "Delta")
cords <- c(1, 4, 5, 2, 5, 4)

#generating SVG string
line_chart <- line_chart(
  data = data,
  x = data$time,
  series = c("Gamma", "Delta"),
  series_labels =c("Gamma inc.", "Delta inc.");
  ser_names = names,
  point_cords = cords,
  interval = "years")

#showing the plot
line_chart %>% SVGrenderer
line_chart_dense 

Line chart with more points then categories on x-axis.

Description

Line chart with more points then categories on x-axis.

Usage

line_chart_dense(data, dates, series, interval = "months")

Arguments

data Data frame in wide format.
dates Name of column in ‘data’ which contains dates or vector of dates.
series Vector of column names in ‘data’ with values of time series.
interval Granularity of x axis. One of c(‘weeks’, ‘months’, ‘quarters’, ‘years’). Default value is ‘months’.

Value

SVG string containing chart

Examples

df <- data.frame(
  x = seq.Date(as.Date('2021-01-01'), as.Date('2021-07-01'), length.out = 200),
  'Company_sin' = 5 * sin(seq(
    from = 0,
    to = 2 * pi,
    length.out = 200
  )) + rnorm(200, mean = 5, sd = 0.5),
  'Company_cos' = 5 * cos(seq(
    from = 0,
    to = 2 * pi,
    length.out = 200
  )) + rnorm(200, mean = 5, sd = 0.5))

df <- head(df, n = 199)

line_chart_dense(
  df,
  dates = 'x',
  series = c('Company_sin', 'Company_cos')) %>%
  SVGRenderer()
**line_chart_dense_custom**

*More customizable version of 'line_chart_dense'. User can choose the points to highlight.*

---

**Description**

More customizable version of 'line_chart_dense'. User can choose the points to highlight.

**Usage**

```r
line_chart_dense_custom(
  list,
  vector_x,
  vector_y,
  vector_cat,
  series_labels,
  df_numbers = NULL,
  point_cords = NULL,
  interval = "months"
)
```

**Arguments**

- **list**: list of data frames, each representing one series. Data frame should consist of columns: * containing numeric values from 0 to 100 defining the percentage of distance in one time interval of the point (x - coordinates of the point) * containing the value of a point (y - coordinates of the point) * containing the time interval of the value
- **vector_x**: vector containing the names of columns with x - coordinates of the point in the data frames
- **vector_y**: vector containing the names of columns with y - coordinates of the point in the data frames
- **vector_cat**: vector containing the names of columns with time interval of the point in the data frames
- **series_labels**: vector containing names of series to be shown on the plot
- **df_numbers**: vector containing index of data frame in the list of a value to be marked
- **point_cords**: vector of the same length as df_numbers containing numerical values of indexes in data frame of values to be marked
- **interval**: intervals on x axis. The width of the bars depends on this parameter

**Value**

SVG string containing chart
Examples

```r
# preparing data frames
data <- data.frame(
  xdata = c(1, 60, 90, 30, 60, 90, 30, 60, 90, 45, 95, 45, 95),
  ydata = c(5, -10, -15, 11, 16, 18, 25, 22, 18, 10, 8, 23, 28),
)

df <- data.frame(
  xdf = c(1, 60, 90, 30, 60, 90, 30, 60, 90, 45, 95, 45, 95),
  ydf = c(25, 22, 20, 18, 28, 35, 33, 29, 30, 38, 31, 26, 22),
)

# defining the rest of the arguments
list <- list(data, df)
vector_x <- c("xdata", "xdf")
vector_y <- c("ydata", "ydf")
vector_cat <- c("catdata", "catdf")
df_numbers <- c(1, 2, 2, 1)
point_cords <- c(1, 3, 4, 10)

# generating the SVG string
plot <- line_chart_dense_custom(
  list,
  vector_x = c("xdata", "xdf"),
  vector_y = c("ydata", "ydf"),
  vector_cat = c("catdata", "catdf"),
  series_labels = c("Gamma inc.", "Delta inc."),
  df_numbers = df_numbers,
  point_cords = point_cords)

# showing the plot
plot %>% SVGrenderer()
```

Description

Generates line plot with markers on every value.
Usage

```r
line_chart_markers(
  data,
  x,
  series,
  series_labels,
  interval = "months",
  styles = NULL
)
```

Arguments

data: data frame containing data to be plotted
x: vector containing time intervals of the values
series: vector containing names of columns in data with values to plot
series_labels: vector containing names of series to be shown on the plot
interval: intervals on x axis. The width of the bars depends on this parameter
styles: optional data frame with style names. Styles of the markers will be plotted accordingly.

Value

SVG string containing chart

Examples

```r
# preparing a data frame
data <- data.frame(
  time = c("Jan", "Feb", "Mar", "Apr", "May", "Jun"),
  PL = (c(51, 42, 50, 58, 78, 79) - 30),
  AC = (c(62, 70, 67, 77, 63, 62) - 30)
)
# preparing the styles data frame
styles <- data.frame(
  PL = c("plan", "plan", "plan", "plan", "plan", "plan"),
  AC = c("actual", "actual", "actual", "forecast", "forecast", "forecast")
)
# generating svg string
line_chart <- line_chart_markers(data, data$time, c("PL", "AC"), c("PL", "AC"), "months", styles)
# show the plot
line_chart %>% SVGrenderer()
```
Generates line plot with markers on every value with index on a given value.

Usage

```r
line_chart_markers_reference(
  data,
  x,
  series,
  series_labels,
  ref_val,
  ref_label = ref_val,
  interval = "months",
  styles = NULL
)
```

Arguments

- `data`: data frame containing data to be plotted
- `x`: vector containing time intervals of the values
- `series`: vector containing names of columns in data with values to plot
- `series_labels`: vector containing names of series to be shown on the plot
- `ref_val`: numeric value of the index
- `ref_label`: string defining a text that should be displayed in the referencing line. Set by default to `index_val`.
- `interval`: intervals on x axis. The width of the bars depends on this parameter
- `styles`: optional data frame with style names. Styles of the markers will be plotted accordingly.

Value

SVG string containing chart

Examples

```r
# preparing a data frame
data <- data.frame(
  time = c("Jan", "Feb", "Mar", "Apr", "May", "Jun"),
  PL = (c(51, 42, 50, 58, 78, 79) - 30),
)
AC = (c(62, 70, 67, 77, 63, 62) - 30)
)
# preparing the styles data frame
styles <- data.frame(
  PL = c("plan", "plan", "plan", "plan", "plan", "plan"),
  AC = c("actual", "actual", "actual", "forecast", "forecast", "forecast")
)

# generating svg string
line_chart_ref <- line_chart_markers_reference(
  data = data,
  x = data$time,
  series = c("PL", "AC"),
  series_labels = c("PL", "AC"),
  ref_val = 42,
  ref_label = "index",
  styles=styles)

# show the plot
line_chart_ref %>% SVGrenderer()

---

**line_chart_normalized**  
*Generates normalized areas (stacked lines) plot. If more than one series is supplied, stacked areas plot is generated.*

**Description**

Generates normalized areas (stacked lines) plot. If more than one series is supplied, stacked areas plot is generated.

**Usage**

```r
line_chart_normalized(
  data,
  x,
  series,
  series_labels,
  show_labels,
  interval = "months"
)
```

**Arguments**

- **data**: data frame containing data to be plotted
- **x**: vector containing time intervals of the values
- **series**: vector containing names of columns in data with values to plot
- **series_labels**: vector containing names of series to be shown on the plot
show_labels vector of the same length as cat containing NA or not NA values defining which categories should have labels of values displayed

interval intervals on x axis. The width of the bars depends on this parameter

Value

SVG SVG string containing chart

Examples

```r
#preparing data frames
data <- data.frame(weeks = c(28, 29, 30, 31, 32, 33, 34, 35, 36, 37), Services = c(130, 150, 182, 170, 170, 140, 130, 130, 135, 140), Software = c(100, 88, 83, 90, 92, 95, 129, 130, 130, 135), Products = c(20, 35, 36, 40, 22, 25, 24, 19, 36, 40) )

# defining the rest of the arguments
series <- c("Software", "Services", "Products")
labels <- c(NA, 1, NA, 1, NA, 1, NA, 1, NA)

# generating the SVG string
line_chart_normalized <- line_chart_normalized(data, data$weeks, series, series, labels, "weeks")

# show the plot
line_chart_normalized %>% SVGrenderer()
```

**line_chart_stacked** Generates areas (stacked lines) plot. If more than one series is supplied, stacked areas plot is generated.

Description

Generates areas (stacked lines) plot. If more than one series is supplied, stacked areas plot is generated.

Usage

```r
line_chart_stacked( data, cat, series, series_labels, show_labels, interval = "months"
)```
parse_time_series

Function to transfer data frame with time series values in wide format to format accepted by 'line_chart_dense_custom'.

Description

Function to transfer data frame with time series values in wide format to format accepted by 'line_chart_dense_custom'.

Usage

parse_time_series(df, dates, series, convert.to = "months")

Arguments

data: data frame containing data to be plotted

cat: vector containing time intervals of the values

series: vector containing names of columns in data with values to plot

series_labels: vector containing names of series to be shown on the plot

show_labels: vector of the same length as cat containing NA or not NA values defining which categories should have labels of values displayed

interval: intervals on x axis. The width of the bars depends on this parameter

Value

SVG string containing chart

Examples

#preparing data frames
data <- data.frame(weeks = c(28, 29, 30, 31, 32, 33, 34, 35, 36, 37), Services = c(130,150, 182, 170, 140, 130, 135, 140), Software = c(100, 88, 83, 90, 92, 95, 129, 130, 130, 135), Products = c(20, 35, 36, 40, 22, 25, 24, 19, 36, 40) )

#defining the rest of the arguments
series <- c("Software", "Services", "Products")
labels <- c(NA, 1, NA, 1, NA, NA, 1, NA, 1, NA)

#generating the SVG string
line_chart_stacked <- line_chart_stacked(data, data$weeks, series, series, labels, "weeks")

#show the plot
line_chart_stacked %>% SVGrenderer()
### Arguments

- **df**: Date frame with data in wide format.
- **dates**: Name of column in `df` which contains dates.
- **series**: Vector of column names in `df` with values of time series.
- **convert.to**: Granularity of x axis. One of c(‘weeks’, ‘months’, ‘quarters’, ‘years’). Default value is ‘months’.

### Value

List of data frames, each one containing data about one time series. Data frames in returned list consist of columns:
- containing numeric values from 0 to 100 defining the percentage of distance in one time interval of the point (x - coordinates of the point)
- containing the value of a point (y - coordinates of the point)
- containing the time interval of the value

### Examples

```r
df <- data.frame(
  dates = as.Date(c('2021-07-12', '2021-06-18', '2021-05-12'))),
  val1 = c(1.5, 1.2, 2.1),
  val2 = c(0.9, 3.2, 1.1))
parse_time_series(df, 'dates', c('val1', 'val2'))
```

---

### restore_defaults

*Restore default color and style settings*

### Description

Restore default color and style settings

### Usage

```r
restore_defaults()
```

### Value

No return value, called for side effects

### Examples

```r
restore_defaults()
```
scatter_plot  

Generates a scatter plot. If additional argument added, a bubble plot is generated.

**Description**

Generates a scatter plot. If additional argument added, a bubble plot is generated.

**Usage**

```r
scatter_plot(
  data,
  x,
  y,
  cat = NULL,
  x_space_size = (x_end - x_start)/8,
  y_space_size = (y_end - y_start)/8,
  x_names = c("x", ""),
  y_names = c("y", ""),
  legend_title = "Legend",
  bubble_value = NULL,
  x_start = 0,
  x_end = max(x),
  y_start = 0,
  y_end = max(y)
)
```

**Arguments**

- **data**: data frame containing data to be plotted
- **x**: string containing a column name or a vector containing x - coordinates of values
- **y**: string containing a column name or a vector containing y - coordinates of values
- **cat**: string containing a column name or a vector containing categories of the values
- **x_space_size, y_space_size**: numeric value of the space between the ticks on the x,y - axis. Defaultly, axis will be divided into 8 sections
- **x_names**: vector containing two values: * name of the value presented on the x - axis * units of values presented on the x - axis
- **y_names**: vector containing two values: * name of the value presented on the y - axis * units of values presented on the y - axis
- **legend_title**: title of the legend
- **bubble_value**: vector containing values defining the size of bubbles. Set by default to NULL.
- **x_start**: numeric value defining where the x axis should start at. Set by default to 0.
- **x_end**: numeric value defining where the x axis should end at. Set by default to max(x).
- **y_start**: numeric value defining where the y axis should start at. Set by default to 0.
- **y_end**: numeric value defining where the y axis should end at. Set by default to max(y).
Value

SVG string containing chart

Examples

```r
# prepare a data frame
data <- data.frame(
  x = c(2, -3, -5, 5.5, 7, 9, 2.5, 1, 5, 5.3, 8.5, 6.6),
  value = c(5,-3,2,6, 7, -2, 1,7,8,3, -5),
  cat = c("val1","val1","val2","val2", "val3","val3","val4","val4","val4","val4"),
  bubble = c (1,2,4,5,4,8,2,1,9, 8, 4.5 )
)

# generate character vectors with svg data
scatter <- scatter_plot(
  data = data,
  x = data$x,
  y = data$value,
  cat = data$cat,
  x_space_size = 2,
  y_space_size = 1,
  x_names = c("time", "in s"),
  y_names = c("distance", "in km"),
  legend_title = "Legend"
)

bubble <- scatter_plot(
  data = data,
  x = data$x,
  y = data$value,
  cat = data$cat,
  x_space_size = 2,
  y_space_size = 1,
  x_names = c("time", "in s"),
  y_names = c("distance", "in km"),
  legend_title = "Legend",
  bubble_value = data$bubble)

# show the plots
scatter %>% SVGrenderer()
bubble %>% SVGrenderer()
```

---

**set_colors**

Change default colors of the package

**Description**

Customize your plots and change default color palette.
set_styles

Usage

set_colors(colors_df)

Arguments

colors_df  data frame with 6 rows and 2 columns. Columns must have names: "text_colors", "bar_colors". In cells there should be rgb values of chosen colors in format: "rgb(x,y,z)". Rows represent subsequent colors on stacked plots.

Value

No return value, called for side effects

Examples

mi2lab_colors <- cbind(
  bar_colors = c(
    "rgb(68, 19, 71)",
    "rgb(243, 46, 255)",
    "rgb(106, 0, 112)",
    "rgb(217, 43, 227)",
    "rgb(114, 49, 117)",
    "rgb(249, 110, 255)"
  ),
  text_colors = c("white", "white", "white", "white", "white", "white")
)

set_colors(mi2lab_colors)

set_styles

Change default styles for plots

Description

Change default styles for plots

Usage

set_styles(styles_df)

Arguments

styles_df  data frame with columns 'fill' and 'stroke'. Rows represent subsequent styles which names can be passed to plotting functions, usually as styles argument.

Value

No return value, called for side effects
Examples

```r
styles_df <-
  rbind(
    actual = c("rgb(64,64,64)", "rgb(64,64,64)"),
    previous =
      c("rgb(166,166,166)", "rgb(166,166,166)"),
    forecast =
      c("url(#diagonalHatch)", "rgb(64,64,64)"),
    plan = c("white", "rgb(64,64,64)"),
    total_white = c("white", "white")
  )
colnames(styles_df) <- c("fill", "stroke")
set_styles(styles_df)
```

---

**str_width**

_Calculate string width in pixels_

**Description**

Calculate string width in pixels

**Usage**

```r
str_width(string, bold = FALSE)
```

**Arguments**

- `string`: string which width will be calculated
- `bold`: boolean value, if string will be written in bold

**Value**

- string width in pixels

---

**SVGrenderer**

_Function to render SVG image as htmlwidget_

**Description**

Use this function to show SVG images from given string in SVG embedded in HTML.

**Usage**

```r
SVGrenderer(svg_string, width = NULL, height = NULL, elementId = NULL)
```
Arguments

- `svg_string`: one element character vector with image in svg format
- `width`: width of the widget
- `height`: height of the widget
- `elementId`: HTML element ID

Value

No return value, called for side effects

---

**SVGsave**  
*Save svg image*

Description

Save svg image

Usage

```r
SVGsave(svg_string, path)
```

Arguments

- `svg_string`: string containing SVG statements
- `path`: path to file where image will be saved

Value

No return value, called for side effects
Index

add_bars, 3
add_title, 4
add_waterfall_bars, 5

bar_chart, 6
bar_chart_absolute_variance, 7
bar_chart_grouped, 8
bar_chart_normalized, 9
bar_chart_reference, 10
bar_chart_relative_variance, 11
bar_chart_waterfall, 12

column_chart, 13
column_chart_absolute_variance, 14
column_chart_grouped, 15
column_chart_normalized, 16
column_chart_reference, 17
column_chart_relative_variance, 19
column_chart_waterfall, 20
column_chart_waterfall_variance, 21

draw_triangle, 22

facet_chart, 22

get_gray_color_stacked, 23
get_vector, 24

join_charts, 24

line_chart, 25
line_chart_dense, 27
line_chart_dense_custom, 28
line_chart_markers, 29
line_chart_markers_reference, 31
line_chart_normalized, 32
line_chart_stacked, 33

parse_time_series, 34

restore_defaults, 35

scatter_plot, 36
set_colors, 37
set_styles, 38
str_width, 39
SVGrenderer, 39
SVGsave, 40