

# Package ‘easyalluvial’

November 18, 2018

**Type** Package

**Title** Generate Alluvial Plots with a Single Line of Code

**Version** 0.1.900

**Author** bjoern koneswarakantha

**Maintainer** bjoern koneswarakantha <datistics@gmail.com>

**Description** Alluvial plots are a form of sankey diagram that are a great tool for visualising categorical data. Their graphical grammar however is a bit more complex than that of a regular x/y plots. The ggalluvial package made a great job of translating that grammar into ggplot2 syntax and gives you many options to tweak the appearance of an alluvial plot, however there still remains a multi-layered complexity that makes it difficult to use ggalluvial for explorative data analysis. easyalluvial provides a simple interface to this package that allows you to put out a decent alluvial from any dataframe where data is stored in either long or wide format while also handling continuous data. It is meant to allow a quick visualisation of entire dataframes similar to the visualisations created by the tabplot package with a focus on different colouring options that can make alluvials a great tool for data exploration.

**License** CC0

**Encoding** UTF-8

**LazyData** true

**Suggests** testthat,  
covr

**RoxygenNote** 6.1.0

**Imports** purrr  
, tidy  
, dplyr  
, forcats  
, ggalluvial  
, ggplot2  
, RColorBrewer  
, recipes  
, nycflights13  
, rlang  
, stringr  
, ISLR  
, magrittr  
, tibble

## R topics documented:

alluvial_long . . . . .	2
alluvial_wide . . . . .	4
manip_bin_numerics . . . . .	6
manip_factor_2_numeric . . . . .	7
palette_filter . . . . .	8
palette_increase_length . . . . .	9
palette_plot_intensity . . . . .	10
palette_plot_rgp . . . . .	10
palette_qualitative . . . . .	11
plot_condensation . . . . .	12

<b>Index</b>	<b>13</b>
--------------	-----------

---

alluvial_long	<i>alluvial plot of data in long format</i>
---------------	---

---

### Description

Plots two variables of a dataframe on an alluvial plot. A third variable can be added either to the left or the right of the alluvial plot to provide coloring of the flows. All numerical variables are scaled, centered and YeoJohnson transformed before binning.

### Usage

```
alluvial_long(data, key, value, id, fill = NULL, fill_right = T,
  bins = 5, bin_labels = c("LL", "ML", "M", "MH", "HH"),
  NA_label = "NA", order_levels_value = NULL,
  order_levels_key = NULL, order_levels_fill = NULL, complete = TRUE,
  fill_by = "first_variable", col_vector_flow = palette_qualitative()
  %>% palette_filter(greys = F),
  col_vector_value = RColorBrewer::brewer.pal(9, "Greys")[c(3, 6, 4, 7,
  5)], verbose = F, stratum_labels = T, stratum_width = 1/4,
  auto_rotate_xlabs = T)
```

### Arguments

data	a dataframe
key	unquoted column name or string of x axis variable
value	unquoted column name or string of y axis variable
id	unquoted column name or string of id column
fill	unquoted column name or string of fill variable which will be used to color flows, Default: NULL
fill_right	logical, TRUE fill variable is added to the right FALSE to the left, Default: T
bins	number of bins for automatic binning of numerical variables, Default: 5
bin_labels	labels for bins, Default: c("LL", "ML", "M", "MH", "HH")
NA_label	character vector define label for missing data

order_levels_value	character vector denoting order of y levels from low to high, does not have to be complete can also just be used to bring levels to the front, Default: NULL
order_levels_key	character vector denoting order of x levels from low to high, does not have to be complete can also just be used to bring levels to the front, Default: NULL
order_levels_fill	character vector denoting order of color fill variable levels from low to high, does not have to be complete can also just be used to bring levels to the front, Default: NULL
complete	logical, insert implicitly missing observations, Default: TRUE
fill_by	one_of(c('first_variable', 'last_variable', 'all_flows', 'values')), Default: 'first_variable'
col_vector_flow	HEX color values for flows, Default: palette_filter( greys = F)
col_vector_value	HEX color values for y levels/values, Default:RColorBrewer::brewer.pal(9, 'Greys')[c(3,6,4,7,5)]
verbose	logical, print plot summary, Default: F
stratum_labels	logical, Default: TRUE
stratum_width	double, Default: 1/4
auto_rotate_xlabs	logical, Default: TRUE

**Value**

ggplot2 object

**See Also**

[alluvial\\_wide](#), [geom\\_flow](#), [geom\\_stratum](#)

**Examples**

```
## Not run:
if(interactive()){

  # sample data-----

  require(magrittr)
  require(dplyr)
  require(tidyr)

  monthly_flights = nycflights13::flights %>%
    group_by(month, tailnum, origin, dest, carrier) %>%
    summarise() %>%
    group_by( tailnum, origin, dest, carrier) %>%
    count() %>%
    filter( n == 12 ) %>%
    select( - n ) %>%
    left_join( nycflights13::flights ) %>%
    .[complete.cases(.), ] %>%
    ungroup() %>%
    mutate( tailnum = pmap_chr(list(tailnum, origin, dest, carrier), paste )
            , qu = cut(month, 4)) %>%
```

```

group_by(tailnum, carrier, origin, dest, qu ) %>%
summarise( mean_arr_delay = mean(arr_delay) ) %>%
ungroup() %>%
mutate( mean_arr_delay = ifelse( mean_arr_delay < 10, 'on_time', 'late' ) )

levels(monthly_flights$qu) = c('Q1', 'Q2', 'Q3', 'Q4')

data = monthly_flights

# flow coloring variants -----
alluvial_long( data, key = qu, value = mean_arr_delay, id = tailnum, fill = carrier )
alluvial_long( data, key = qu, value = mean_arr_delay, id = tailnum, fill_by = 'last_variable' )
alluvial_long( data, key = qu, value = mean_arr_delay, id = tailnum, fill_by = 'first_variable' )
alluvial_long( data, key = qu, value = mean_arr_delay, id = tailnum, fill_by = 'all_flows' )
alluvial_long( data, key = qu, value = mean_arr_delay, id = tailnum, fill_by = 'value' )

# use same color coding for flows and y levels -----
alluvial_long( data, qu, mean_arr_delay, tailnum, fill_by = 'value'
, col_vector_flow = palette_qualitative() %>% palette_filter(greys = F, bright = F)
, col_vector_value = palette_qualitative() %>% palette_filter(greys = F, bright = F) )

# move fill variable to the left -----
alluvial_long( data, qu, mean_arr_delay, tailnum, carrier ,fill_right = F )

# reorder levels -----
alluvial_long( data, qu, mean_arr_delay, tailnum, fill_by = 'first_variable'
, order_levels_value = c('on_time', 'late') )

alluvial_long( data, qu, mean_arr_delay, tailnum, fill_by = 'first_variable'
, order_levels_key = c('Q4', 'Q3', 'Q2', 'Q1') )

order_by_carrier_size = data %>%
  group_by(carrier) %>%
  count() %>%
  arrange( desc(n) ) %>%
  .[['carrier']]

alluvial_long( data, qu, mean_arr_delay, tailnum, carrier
, order_levels_fill = order_by_carrier_size )

}

## End(Not run)

```

---

alluvial\_wide

*alluvial plot of data in wide format*


---

### Description

plots a dataframe as an alluvial plot. All numerical variables are scaled, centered and YeoJohnson transformed before binning. Plots all variables in the sequence as they appear in the dataframe until maximum number of values is reached.

**Usage**

```
alluvial_wide(data, id = NULL, max_variables = 20, bins = 5,
  bin_labels = c("LL", "ML", "M", "MH", "HH"), NA_label = "NA",
  order_levels = NULL, fill_by = "first_variable",
  col_vector_flow = palette_qualitative() %>% palette_filter(greys =
  F), col_vector_value = RColorBrewer::brewer.pal(9, "Greys")[c(4, 7, 5,
  8, 6)], verbose = F, stratum_labels = T, stratum_width = 1/4,
  auto_rotate_xlabs = T)
```

**Arguments**

<code>data</code>	a dataframe
<code>id</code>	unquoted column name of id column or character vector with id column name
<code>max_variables</code>	maximum number of variables, Default: 20
<code>bins</code>	number of bins for numerical variables, Default: 5
<code>bin_labels</code>	labels for the bins from low to high, Default: c("LL", "ML", "M", "MH", "HH")
<code>NA_label</code>	character vector define label for missing data, Default: 'NA'
<code>order_levels</code>	character vector denoting levels to be reordered from low to high
<code>fill_by</code>	one_of(c('first_variable', 'last_variable', 'all_flows', 'values')), Default: 'first_variable'
<code>col_vector_flow</code>	HEX colors for flows, Default: palette_filter( greys = F)
<code>col_vector_value</code>	Hex colors for y levels/values, Default: RColorBrewer::brewer.pal(9, "Greys")[c(3, 6, 4, 7, 5)]
<code>verbose</code>	logical, print plot summary, Default: F
<code>stratum_labels</code>	logical, Default: TRUE
<code>stratum_width</code>	double, Default: 1/4
<code>auto_rotate_xlabs</code>	logical, Default: TRUE

**Details**

Under the hood this function converts the wide format into long format. `ggalluvial` also offers a way to make alluvial plots directly from wide format tables but it does not allow individual colouring of the stratum segments. The tradeoff is that we can only order levels as a whole and not individually by variable, Thus if some variables have levels with the same name the order will be the same. If we want to change level order independently we have to assign unique level names first.

**Value**

ggplot2 object

**See Also**

[alluvial\\_wide](#), [geom\\_flow](#), [geom\\_stratum](#)

**Examples**

```

## Not run:
if(interactive()){

  require(magrittr)
  require(dplyr)

  data = as_tibble(mtcars)
  categoricals = c('cyl', 'vs', 'am', 'gear', 'carb')
  numericals = c('mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec')
  max_variables = 5

  data = data %>%
    mutate_at( vars(categoricals), as.factor )

  alluvial_wide( data = data
                , max_variables = max_variables
                , fill_by = 'first_variable' )

  alluvial_wide( data = data
                , max_variables = max_variables
                , fill_by = 'last_variable' )

  alluvial_wide( data = data
                , max_variables = max_variables
                , fill_by = 'all_flows' )

  alluvial_wide( data = data
                , max_variables = max_variables
                , fill_by = 'first_variable' )

  # manually order variable values

  alluvial_wide( data = data
                , max_variables = max_variables
                , fill_by = 'values'
                , order_levels = c('1', '0') )
}

## End(Not run)

```

---

manip\_bin\_numerics      *bin numerical columns*

---

**Description**

centers, scales and Yeo Johnson transforms numeric variables in a dataframe before binning into n bins of equal range. Outliers based on boxplot stats are capped (set to min or max of boxplot stats).

**Usage**

```
manip_bin_numerics(x, bins = 5, bin_labels = c("LL", "ML", "M", "MH",
"HH"), center = T, scale = T, transform = T)
```

**Arguments**

x	dataframe with numeric variables, or numeric vector
bins	number of bins for numerical variables, Default: 5
bin_labels	labels for the bins from low to high, Default: c("LL", "ML", "M", "MH", "HH")# @param center boolean, Default: T
center	boolean, Default: T
scale	boolean, Default: T
transform	boolean, Default: T

**Value**

dataframe

---

manip\_factor\_2\_numeric

*converts factor to numeric preserving numeric levels and order in character levels.*

---

**Description**

before converting we check whether the levels contain a number, if they do the number will be preserved.

**Usage**

```
manip_factor_2_numeric(vec)
```

**Arguments**

vec	vector
-----	--------

**Value**

vector

**See Also**

[str\\_detect](#)

**Examples**

```
fac_num = factor( c(1,3,8) )
fac_chr = factor( c('foo','bar') )
fac_chr_ordered = factor( c('a','b','c'), ordered = TRUE )

manip_factor_2_numeric( fac_num )
manip_factor_2_numeric( fac_chr )
manip_factor_2_numeric( fac_chr_ordered )
```

---

palette\_filter                      *color filters for any vector of hex color values*

---

## Description

filters are based on rgb values

## Usage

```
palette_filter(palette = palette_qualitative(), similar = F,
  greys = T, reds = T, greens = T, blues = T, dark = T,
  medium = T, bright = T, thresh_similar = 25)
```

## Arguments

palette	any vector with hex color values, Default: palette_qualitative()
similar,	logical, allow similar colours, similar colours are detected using a threshold (thresh_similar), two colours are similar when each value for RGB is within threshold range of the corresponding RGB value of the second colour, Default: F
greys,	logical, allow grey colours, blue == green == blue , Default: T
reds,	logical, allow red colours, blue < 50 & green < 50 & red > 200 , Default: T
greens,	logical, allow green colours, green > red & green > blue, Default: T
blues,	logical, allow blue colours, blue > green & green > red, Default: T
dark,	logical, allow colours of dark intensity, sum( red, green, blue) < 420 , Default: T
medium,	logical, allow colours of medium intensity, between( sum( red, green, blue), 420, 600) , Default: T
bright,	logical, allow colours of bright intensity, sum( red, green, blue) > 600, Default: T
thresh_similar,	int, threshold for defining similar colours, see similar, Default: 25

## Value

vector with hex colors

## Examples

```
## Not run:
if(interactive()){

  require(magrittr)

  palette_qualitative() %>%
    palette_filter(thresh_similar = 0) %>%
    palette_plot_intensity()

  palette_qualitative() %>%
    palette_filter(thresh_similar = 25) %>%
```

```
palette_plot_intensity()

palette_qualitative() %>%
  palette_filter(thresh_similar = 0, blues = FALSE) %>%
  palette_plot_intensity()

}

## End(Not run)
```

---

`palette_increase_length`

*increases length of palette by repeating colours*

---

### **Description**

works for any vector

### **Usage**

```
palette_increase_length(palette = palette_qualitative(), n = 100)
```

### **Arguments**

`palette` any vector, Default: `palette_qualitative()`  
`n`, int, length, Default: 100

### **Value**

vector with increased length

### **Examples**

```
require(magrittr)

length(palette_qualitative())

palette_qualitative() %>%
  palette_increase_length(100) %>%
  length()
```

palette\_plot\_intensity  
*plot colour intensity of palette*

---

**Description**

sum of red green and blue values

**Usage**

```
palette_plot_intensity(palette)
```

**Arguments**

palette            any vector containing color hex values

**Value**

ggplot2 plot

**See Also**

[palette\\_plot\\_rgp](#)

**Examples**

```
## Not run:  
if(interactive()){  
  palette_qualitative() %>%  
    palette_filter( thresh = 25) %>%  
    palette_plot_intensity()  
}  
  
## End(Not run)
```

---

palette\_plot\_rgp        *plot rgb values of palette*

---

**Description**

grouped bar chart

**Usage**

```
palette_plot_rgp(palette)
```

**Arguments**

palette            any vector containing color hex values

**Value**

ggplot2 plot

**See Also**

[palette\\_plot\\_intensity](#)

**Examples**

```
## Not run:  
if(interactive()){  
  palette_qualitative() %>%  
    palette_filter( thresh = 50) %>%  
    palette_plot_rgp()  
}  
  
## End(Not run)
```

---

palette\_qualitative    *compose palette from qualitative RColorBrewer palettes*

---

**Description**

combines all unique values found in all qualitative RColorBrewer palettes

**Usage**

```
palette_qualitative()
```

**Value**

vector with hex values

**See Also**

[RColorBrewer](#)

**Examples**

```
palette_qualitative()
```

---

plot_condensation	<i>Plot dataframe condensation potential</i>
-------------------	--

---

### Description

plotting the condensation potential is meant as a decision aid for which variables to include in an alluvial plot. All variables are transformed to categoric variables and then two variables are selected by which the dataframe will be grouped and summarized by. The pair that results in the greatest condensation of the original dataframe is selected. Then the next variable which offers the greatest condensation potential is chosen until all variables have been added. The condensation in percent is then plotted for each step along with the number of groups (flows) in the dataframe. By experience it is not advisable to have more than 1500 flows because then the alluvial plot will take a long time to render. If there is a particular variable of interest in the dataframe this variable can be chosen as a starting variable.

### Usage

```
plot_condensation(df, first = NULL)
```

### Arguments

df	dataframe
first	unquoted expression or string denoting the first variable to be picked for condensation, Default: NULL

### Value

ggplot2 plot

### See Also

[quosure reexports RColorBrewer](#)

### Examples

```
## Not run:
if(interactive()){
  df = mtcars %>%
    mutate( cyl = as.factor(cyl)
           , gear = as.factor(gear)
           , vs = as.factor(vs)
           , am = as.factor(am))

  plot_condensation(df)

  plot_condensation(df, first = 'disp')
}

## End(Not run)
```

# Index

alluvial\_long, [2](#)  
alluvial\_wide, [3, 4, 5](#)

geom\_flow, [3, 5](#)  
geom\_stratum, [3, 5](#)

manip\_bin\_numerics, [6](#)  
manip\_factor\_2\_numeric, [7](#)

palette\_filter, [8](#)  
palette\_increase\_length, [9](#)  
palette\_plot\_intensity, [10, 11](#)  
palette\_plot\_rgp, [10, 10](#)  
palette\_qualitative, [11](#)  
plot\_condensation, [12](#)

quosure, [12](#)

RColorBrewer, [11, 12](#)  
reexports, [12](#)

str\_detect, [7](#)