

# Package ‘easyalluvial’

May 8, 2026

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

**Version** 0.4.0

**URL** <https://github.com/erblast/easyalluvial/>,  
<https://erblast.github.io/easyalluvial/>

**BugReports** <https://github.com/erblast/easyalluvial/issues>

**Description** Alluvial plots are similar to sankey diagrams and visualise categorical data over multiple dimensions as flows. (Rosvall M, Bergstrom CT (2010) Mapping Change in Large Networks. PLoS ONE 5(1): e8694. <[doi:10.1371/journal.pone.0008694](https://doi.org/10.1371/journal.pone.0008694)> 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 produce a decent alluvial plot from any dataframe in either long or wide format from a single line of code while also handling continuous data. It is meant to allow a quick visualisation of entire dataframes with a focus on different colouring options that can make alluvial plots a great tool for data exploration.

**License** CC0

**Encoding** UTF-8

**LazyData** true

**Depends** R(>= 3.5)

**Suggests** testthat, covr, ISLR, nycflights13, vdiffR (>= 0.3.1),  
pkgdown, mlbench, earth, workflows, future, furr, e1071,  
caret, parsnip, vip, rpart, glmnet, xgboost

**RoxygenNote** 7.3.2

**Imports** purrr , tidyr (>= 1.0.0) , dplyr , forcats , ggalluvial (>= 0.9.1) , ggplot2 (>= 3.2.0) , ggridges , RColorBrewer , recipes (>= 0.1.5) , rlang , stringr , magrittr , tibble , gridExtra , randomForest , progressr , progress

**Language** en-US

**NeedsCompilation** no

**Author** Bjoern Koneswarakantha [aut, cre] (ORCID:  
<<https://orcid.org/0000-0003-4585-7799>>)

**Maintainer** Bjoern Koneswarakantha <datistics@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-09-03 16:20:17 UTC

## Contents

add_imp_plot . . . . .	3
add_marginal_histograms . . . . .	4
alluvial_long . . . . .	5
alluvial_model_response . . . . .	8
alluvial_model_response_caret . . . . .	10
alluvial_model_response_parsnip . . . . .	12
alluvial_wide . . . . .	14
check_pkg_installed . . . . .	17
get_data_space . . . . .	17
get_pdp_predictions . . . . .	19
get_pdp_predictions_seq . . . . .	21
manip_bin_numerics . . . . .	22
manip_factor_2_numeric . . . . .	23
manip_get_ggplot_data . . . . .	24
mtcars2 . . . . .	24
palette_filter . . . . .	25
palette_increase_length . . . . .	26
palette_plot_intensity . . . . .	27
palette_plot_rgp . . . . .	28
palette_qualitative . . . . .	28
plot_all_hists . . . . .	29
plot_condensation . . . . .	30
plot_hist . . . . .	31
plot_imp . . . . .	31
quarterly_flights . . . . .	32
quarterly_sunspots . . . . .	33
tidy_imp . . . . .	33
titanic . . . . .	34

**Index**

**35**

---

add_imp_plot	<i>add bar plot of important features to model response alluvial plot</i>
--------------	---

---

**Description**

adds bar plot of important features to model response alluvial plot

**Usage**

```
add_imp_plot(grid, p = NULL, data_input, plot = T, ...)
```

**Arguments**

grid	gtable or ggplot
p	alluvial plot, optional if alluvial plot has already been passed as grid. Default: NULL
data_input	dataframe used to generate alluvial plot
plot	logical if plot should be drawn or not
...	additional parameters passed to <a href="#">plot_imp</a>

**Value**

gtable

**See Also**

[arrangeGrob](#) [plot\\_imp](#)

**Examples**

```
## Not run:
df = mtcars2[, ! names(mtcars2) %in% 'ids' ]

train = caret::train( disp ~ .
                      , df
                      , method = 'rf'
                      , trControl = caret::trainControl( method = 'none' )
                      , importance = TRUE )

pred_train = caret::predict.train(train, df)

p = alluvial_model_response_caret(train, degree = 4, pred_train = pred_train)

p_grid = add_marginal_histograms(p, data_input = df)

p_grid = add_imp_plot(p_grid, p, data_input = df)

## End(Not run)
```

---

```
add_marginal_histograms
```

*add marginal histograms to alluvial plot*

---

### Description

will add density histograms and frequency plots of original data to alluvial plot

### Usage

```
add_marginal_histograms(
  p,
  data_input,
  top = TRUE,
  keep_labels = FALSE,
  plot = TRUE,
  ...
)
```

### Arguments

<code>p</code>	alluvial plot
<code>data_input</code>	dataframe, input data that was used to create dataframe
<code>top</code>	logical, position of histograms, if FALSE adds them at the bottom, Default: TRUE
<code>keep_labels</code>	logical, keep title and caption, Default: FALSE
<code>plot</code>	logical if plot should be drawn or not
<code>...</code>	additional arguments for model response alluvial plot concerning the response variable
	<b>pred_train</b> display training prediction, not necessary if <code>pred_train</code> has already been passed to <code>alluvial_model_response()</code>
	<b>scale</b> int, y-axis distance between the ridge plots, Default: 400
	<b>resp_var</b> character vector, specify response variable in <code>data_input</code> , if not set response variable will try to be inferred, Default: NULL

### Value

gtable

### See Also

[arrangeGrob](#)

## Examples

```
## Not run:
p = alluvial_wide(mtcars2, max_variables = 3)
p_grid = add_marginal_histograms(p, mtcars2)

## End(Not run)
```

---

alluvial\_long

*alluvial plot of data in long format*

---

## 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_label_type = "label",
  stratum_label_size = 4.5,
  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_label_type	character, Default: "label"
stratum_label_size	numeric, Default: 4.5
stratum_width	double, Default: 1/4
auto_rotate_xlabs	logical, Default: TRUE
...	additional parameter passed to <a href="#">manip_bin_numerics</a>

**Value**

ggplot2 object

**See Also**

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

**Examples**

```
## Not run:
data = quarterly_flights

alluvial_long( data, key = qu, value = mean_arr_delay, id = tailnum, fill_by = 'last_variable' )

# more flow coloring variants -----

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' )

# color by additional variable carrier -----

alluvial_long( data, key = qu, value = mean_arr_delay, fill = carrier, id = tailnum )

# use same color coding for flows and y levels -----

palette = c('green3', 'tomato')

alluvial_long( data, qu, mean_arr_delay, tailnum, fill_by = 'value'
              , col_vector_flow = palette
              , col_vector_value = palette )

# 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') )

require(dplyr)
require(magrittr)

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_model_response
      create model response plot
```

---

### Description

alluvial plots are capable of displaying higher dimensional data on a plane, thus lend themselves to plot the response of a statistical model to changes in the input data across multiple dimensions. The practical limit here is 4 dimensions. We need the data space (a sensible range of data calculated based on the importance of the explanatory variables of the model as created by [get\\_data\\_space](#) and the predictions returned by the model in response to the data space.

### Usage

```
alluvial_model_response(
  pred,
  dspace,
  imp,
  degree = 4,
  bin_labels = c("LL", "ML", "M", "MH", "HH"),
  col_vector_flow = c("#FF0065", "#009850", "#A56F2B", "#005EAA", "#710500", "#7B5380",
    "#9DD1D1"),
  method = "median",
  force = FALSE,
  params_bin_numeric_pred = list(bins = 5),
  pred_train = NULL,
  stratum_label_size = 3.5,
  ...
)
```

### Arguments

pred	vector, predictions, if method = 'pdp' use <a href="#">get_pdp_predictions</a> to calculate predictions
dspace	data frame, returned by <a href="#">get_data_space</a>
imp	dataframe, with not more than two columns one of them numeric containing importance measures and one character or factor column containing corresponding variable names as found in training data.
degree	integer, number of top important variables to select. For plotting more than 4 will result in too many flows and the alluvial plot will not be very readable, Default: 4
bin_labels	labels for prediction bins from low to high, Default: c("LL", "ML", "M", "MH", "HH")
col_vector_flow	character vector, defines flow colours, Default: c('#FF0065', '#009850', '#A56F2B', '#005EAA', '#710500')

method	character vector, one of c('median', 'pdp') <b>median</b> sets variables that are not displayed to median mode, use with regular predictions <b>pdp</b> partial dependency plot method, for each observation in the training data the displayed variable as are set to the indicated values. The predict function is called for each modified observation and the result is averaged, calculate predictions using <a href="#">get_pdp_predictions</a> . Default: 'median'
force	logical, force plotting of over 1500 flows, Default: FALSE
params_bin_numeric_pred	list, additional parameters passed to <a href="#">manip_bin_numerics</a> which is applied to the pred parameter. Default: list( bins = 5, center = T, transform = T, scale = T)
pred_train	numeric vector, base the automated binning of the pred vector on the distribution of the training predictions. This is useful if marginal histograms are added to the plot later. Default = NULL
stratum_label_size	numeric, Default: 3.5
...	additional parameters passed to <a href="#">alluvial_wide</a>

## Details

this model visualisation approach follows the "visualising the model in the dataspace" principle as described in Wickham H, Cook D, Hofmann H (2015) Visualizing statistical models: Removing the blindfold. Statistical Analysis and Data Mining 8(4) <doi:10.1002/sam.11271>

## Value

ggplot2 object

## See Also

[alluvial\\_wide](#), [get\\_data\\_space](#), [alluvial\\_model\\_response\\_caret](#)

## Examples

```
df = mtcars2[, ! names(mtcars2) %in% 'ids' ]
m = randomForest::randomForest( disp ~ ., df)
imp = m$importance
dspace = get_data_space(df, imp, degree = 3)
pred = predict(m, newdata = dspace)
alluvial_model_response(pred, dspace, imp, degree = 3)

# partial dependency plotting method
## Not run:
pred = get_pdp_predictions(df, imp
                           , .f_predict = randomForest::predict.randomForest
                           , m
                           , degree = 3
```

```

, bins = 5)

alluvial_model_response(pred, dspace, imp, degree = 3, method = 'pdp')

## End(Not run)

```

---

```

alluvial_model_response_caret
  create model response plot for caret models

```

---

## Description

Wraps [alluvial\\_model\\_response](#) and [get\\_data\\_space](#) into one call for caret models.

## Usage

```

alluvial_model_response_caret(
  train,
  data_input,
  degree = 4,
  bins = 5,
  bin_labels = c("LL", "ML", "M", "MH", "HH"),
  col_vector_flow = c("#FF0065", "#009850", "#A56F2B", "#005EAA", "#710500", "#7B5380",
    "#9DD1D1"),
  method = "median",
  parallel = FALSE,
  params_bin_numeric_pred = list(bins = 5),
  pred_train = NULL,
  stratum_label_size = 3.5,
  force = F,
  resp_var = NULL,
  ...
)

```

## Arguments

train	caret train object
data_input	dataframe, input data
degree	integer, number of top important variables to select. For plotting more than 4 will result in two many flows and the alluvial plot will not be very readable, Default: 4
bins	integer, number of bins for numeric variables, increasing this number might result in too many flows, Default: 5
bin_labels	labels for the bins from low to high, Default: c("LL", "ML", "M", "MH", "HH")

col_vector_flow	character vector, defines flow colours, Default: c('#FF0065', '#009850', '#A56F2B', '#005EAA', '#710500')
method	character vector, one of c('median', 'pdp')  <b>median</b> sets variables that are not displayed to median mode, use with regular predictions  <b>pdp</b> partial dependency plot method, for each observation in the training data the displayed variables are set to the indicated values. The predict function is called for each modified observation and the result is averaged  . Default: 'median'
parallel	logical, turn on parallel processing for pdp method. Default: FALSE
params_bin_numeric_pred	list, additional parameters passed to <a href="#">manip_bin_numerics</a> which is applied to the pred parameter. Default: list(bins = 5, center = T, transform = T, scale = T)
pred_train	numeric vector, base the automated binning of the pred vector on the distribution of the training predictions. This is useful if marginal histograms are added to the plot later. Default = NULL
stratum_label_size	numeric, Default: 3.5
force	logical, force plotting of over 1500 flows, Default: FALSE
resp_var	character, sometimes target variable cannot be inferred and needs to be passed. Default NULL
...	additional parameters passed to <a href="#">alluvial_wide</a>

### Details

this model visualisation approach follows the "visualising the model in the dataspace" principle as described in Wickham H, Cook D, Hofmann H (2015) Visualizing statistical models: Removing the blindfold. Statistical Analysis and Data Mining 8(4) <doi:10.1002/sam.11271>

### Value

ggplot2 object

### Parallel Processing

We are using 'furr' and the 'future' package to paralelize some of the computational steps for calculating the predictions. It is up to the user to register a compatible backend (see [plan](#)).

### See Also

[alluvial\\_wide](#), [get\\_data\\_space](#), [varImp](#), [extractPrediction](#), [get\\_data\\_space](#), [get\\_pdp\\_predictions](#)

**Examples**

```

if(check_pkg_installed("caret", raise_error = FALSE)) {
  df = mtcars2[, ! names(mtcars2) %in% 'ids' ]

  train = caret::train( disp ~ .,
                        df,
                        method = 'rf',
                        trControl = caret::trainControl( method = 'none' ),
                        importance = TRUE )

  alluvial_model_response_caret(train, df, degree = 3)
}
# partial dependency plotting method
## Not run:
future::plan("multisession")
alluvial_model_response_caret(train, df, degree = 3, method = 'pdp', parallel = TRUE)

## End(Not run)

```

---

```

alluvial_model_response_parsnip
  create model response plot for parsnip models

```

---

**Description**

Wraps [alluvial\\_model\\_response](#) and [get\\_data\\_space](#) into one call for parsnip models.

**Usage**

```

alluvial_model_response_parsnip(
  m,
  data_input,
  degree = 4,
  bins = 5,
  bin_labels = c("LL", "ML", "M", "MH", "HH"),
  col_vector_flow = c("#FF0065", "#009850", "#A56F2B", "#005EAA", "#710500", "#7B5380",
    "#9DD1D1"),
  method = "median",
  parallel = FALSE,
  params_bin_numeric_pred = list(bins = 5),
  pred_train = NULL,
  stratum_label_size = 3.5,
  force = F,
  resp_var = NULL,
  .f_imp = vip::vi_model,
  ...
)

```

**Arguments**

<code>m</code>	parsnip model or trained workflow
<code>data_input</code>	dataframe, input data
<code>degree</code>	integer, number of top important variables to select. For plotting more than 4 will result in too many flows and the alluvial plot will not be very readable, Default: 4
<code>bins</code>	integer, number of bins for numeric variables, increasing this number might result in too many flows, Default: 5
<code>bin_labels</code>	labels for the bins from low to high, Default: c("LL", "ML", "M", "MH", "HH")
<code>col_vector_flow</code>	character vector, defines flow colours, Default: c('#FF0065', '#009850', '#A56F2B', '#005EAA', '#710500')
<code>method</code>	character vector, one of c('median', 'pdp')
	<b>median</b> sets variables that are not displayed to median mode, use with regular predictions
	<b>pdp</b> partial dependency plot method, for each observation in the training data the displayed variables are set to the indicated values. The predict function is called for each modified observation and the result is averaged
	. Default: 'median'
<code>parallel</code>	logical, turn on parallel processing for pdp method. Default: FALSE
<code>params_bin_numeric_pred</code>	list, additional parameters passed to <code>manip_bin_numerics</code> which is applied to the pred parameter. Default: list(bins = 5, center = T, transform = T, scale = T)
<code>pred_train</code>	numeric vector, base the automated binning of the pred vector on the distribution of the training predictions. This is useful if marginal histograms are added to the plot later. Default = NULL
<code>stratum_label_size</code>	numeric, Default: 3.5
<code>force</code>	logical, force plotting of over 1500 flows, Default: FALSE
<code>resp_var</code>	character, sometimes target variable cannot be inferred and needs to be passed. Default NULL
<code>.f_imp</code>	vip function that calculates feature importance, Default: vip::vi_model
<code>...</code>	additional parameters passed to <code>alluvial_wide</code>

**Details**

this model visualisation approach follows the "visualising the model in the dataspace" principle as described in Wickham H, Cook D, Hofmann H (2015) Visualizing statistical models: Removing the blindfold. *Statistical Analysis and Data Mining* 8(4) <doi:10.1002/sam.11271>

**Value**

ggplot2 object

## Parallel Processing

We are using ‘furr’ and the ‘future’ package to parallelize some of the computational steps for calculating the predictions. It is up to the user to register a compatible backend (see [plan](#)).

## See Also

[alluvial\\_wide](#), [get\\_data\\_space](#), [varImp](#), [extractPrediction](#), [get\\_data\\_space](#), [get\\_pdp\\_predictions](#)

## Examples

```
if(check_pkg_installed("parsnip", raise_error = FALSE) &
  check_pkg_installed("vip", raise_error = FALSE)) {
  df = mtcars2[, ! names(mtcars2) %in% 'ids' ]

  m = parsnip::rand_forest(mode = "regression") %>%
    parsnip::set_engine("randomForest") %>%
    parsnip::fit(displ ~ ., data = df)

  alluvial_model_response_parsnip(m, df, degree = 3)
}
## Not run:
# workflow -----
m <- parsnip::rand_forest(mode = "regression") %>%
  parsnip::set_engine("randomForest")

rec_prep = recipes::recipe(displ ~ ., df)

wf <- workflows::workflow() %>%
  workflows::add_model(m) %>%
  workflows::add_recipe(rec_prep) %>%
  parsnip::fit(df)

alluvial_model_response_parsnip(wf, df, degree = 3)

# partial dependence plotting method -----
future::plan("multisession")
alluvial_model_response_parsnip(m, df, degree = 3, method = 'pdp', parallel = TRUE)

## 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)],
  colorful_fill_variable_stratum = T,
  verbose = F,
  stratum_labels = T,
  stratum_label_type = "label",
  stratum_label_size = 4.5,
  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>colorful_fill_variable_stratum</code>	logical, use flow colors to colorize fill variable stratum, Default: TRUE
<code>verbose</code>	logical, print plot summary, Default: F
<code>stratum_labels</code>	logical, Default: TRUE
<code>stratum_label_type</code>	character, Default: "label"
<code>stratum_label_size</code>	numeric, Default: 4.5

```

stratum_width  double, Default: 1/4
auto_rotate_xlabs
                logical, Default: TRUE
...           additional arguments passed to manip\_bin\_numerics

```

### 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](#), [manip\\_bin\\_numerics](#)

### Examples

```

## Not run:
alluvial_wide( data = mtcars2, id = ids
              , max_variables = 3
              , fill_by = 'first_variable' )#'
# more coloring variants-----
alluvial_wide( data = mtcars2, id = ids
              , max_variables = 5
              , fill_by = 'last_variable' )

alluvial_wide( data = mtcars2, id = ids
              , max_variables = 5
              , fill_by = 'all_flows' )

alluvial_wide( data = mtcars2, id = ids
              , max_variables = 5
              , fill_by = 'first_variable' )

# manually order variable values and colour by stratum value

alluvial_wide( data = mtcars2, id = ids
              , max_variables = 5
              , fill_by = 'values'
              , order_levels = c('4', '8', '6') )

## End(Not run)

```

---

check\_pkg\_installed    *check if package is installed*

---

**Description**

check if package is installed

**Usage**

```
check_pkg_installed(pkg, raise_error = TRUE)
```

**Arguments**

pkg                    character, package name  
raise\_error            logical

**Value**

logical

**Examples**

```
check_pkg_installed("easyalluvial")
```

---

get\_data\_space            *calculate data space*

---

**Description**

calculates a dataspace based on the modeling dataframe and the importance of the explanatory variables. It only considers the most important variables as defined by the degree parameter. It selects a number (defined by bins) of sensible single values spread over the range of the numeric variables and creates all possible value combinations among the most important variables. The values of the remaining variables are set to mode(factors) or median( numerics).

**Usage**

```
get_data_space(df, imp, degree = 4, bins = 5, max_levels = 10)
```

**Arguments**

df	dataframe, training data
imp	dataframe, with not more then two columns one of them numeric containing importance measures and one character or factor column containing corresponding variable names as found in training data.
degree	integer, number of top important variables to select. For plotting more than 4 will result in two many flows and the alluvial plot will not be very readable, Default: 4
bins	integer, number of bins for numeric variables, and maximum number of levels for factor variables, increasing this number might result in too many flows, Default: 5
max_levels	integer, maximum number of levels per factor variable, Default: 10

**Details**

It selects a the top most important variables based on the degree parameter and bins the numeric variables using [manip\\_bin\\_numerics](#), while leaving categoric variables unchanged. The number of bins for each numeric variable is set to bins -2. Next the median is picked for each of the bins and the min and the max value is added for each numeric variable So that we get (median(bin) X bins -2, max, min) for each numeric variable. Then all possible combinations between those values and the categoric factor levels are created. The total number of all possible combinations defines the range of the data space. The values of the remaining variables are set to mode(factors) or median( numerics).

this model visualisation approach follows the "visualising the model in the dataspace" principle as described in Wickham H, Cook D, Hofmann H (2015) Visualizing statistical models: Removing the blindfold. Statistical Analysis and Data Mining 8(4) <doi:10.1002/sam.11271>

**Value**

data frame

**See Also**

[alluvial\\_wide](#), [manip\\_bin\\_numerics](#)

**Examples**

```
df = mtcars2[, ! names(mtcars2) %in% 'ids' ]
m = randomForest::randomForest( disp ~ ., df)
imp = m$importance
dspace = get_data_space(df, imp)
```

---

get\_pdp\_predictions *get predictions compatible with the partial dependence plotting method*

---

### Description

Alluvial plots are capable of displaying higher dimensional data on a plane, thus lend themselves to plot the response of a statistical model to changes in the input data across multiple dimensions. The practical limit here is 4 dimensions while conventional partial dependence plots are limited to 2 dimensions.

Briefly the 4 variables with the highest feature importance for a given model are selected and 5 values spread over the variable range are selected for each. Then a grid of all possible combinations is created. All non-plotted variables are set to the values found in the first row of the training data set. Using this artificial data space model predictions are being generated. This process is then repeated for each row in the training data set and the overall model response is averaged in the end. Each of the possible combinations is plotted as a flow which is coloured by the bin corresponding to the average model response generated by that particular combination.

### Usage

```
get_pdp_predictions(
  df,
  imp,
  m,
  degree = 4,
  bins = 5,
  .f_predict = predict,
  parallel = FALSE
)
```

### Arguments

df	dataframe, training data
imp	dataframe, with not more than two columns one of them numeric containing importance measures and one character or factor column containing corresponding variable names as found in training data.
m	model object
degree	integer, number of top important variables to select. For plotting more than 4 will result in too many flows and the alluvial plot will not be very readable, Default: 4
bins	integer, number of bins for numeric variables, increasing this number might result in too many flows, Default: 5
.f_predict	corresponding model predict() function. Needs to accept 'm' as the first parameter and use the 'newdata' parameter. Supply a wrapper for predict functions with x-y syntax. For parallel processing the predict method of object

classes will not always get imported correctly to the worker environment. We can pass the correct predict method via this parameter for example `randomForest::predict.randomForest`. Note that a lot of modeling packages do not export the predict method explicitly and it can only be found using `:::`.

`parallel` logical, turn on parallel processing. Default: FALSE

## Details

For more on partial dependency plots see [<https://christophm.github.io/interpretable-ml-book/pdp.html>].

## Value

vector, predictions

## Parallel Processing

We are using ‘furr’ and the ‘future’ package to paralelize some of the computational steps for calculating the predictions. It is up to the user to register a compatible backend (see [plan](#)).

## Examples

```
df = mtcars2[, ! names(mtcars2) %in% 'ids' ]
m = randomForest::randomForest( disp ~ ., df)
imp = m$importance

pred = get_pdp_predictions(df, imp
                          , m
                          , degree = 3
                          , bins = 5)

# parallel processing -----
## Not run:
future::plan("multisession")

# note that we have to pass the predict method via .f_predict otherwise
# it will not be available in the worker's environment.

pred = get_pdp_predictions(df, imp
                          , m
                          , degree = 3
                          , bins = 5,
                          , parallel = TRUE
                          , .f_predict = randomForest::predict.randomForest)

## End(Not run)
```

---

`get_pdp_predictions_seq`

*get predictions compatible with the partial dependence plotting method, sequential variant that only works for numeric predictions.*

---

### Description

has been replaced by `pdp_predictions` which can be paralelized and also handles factor predictions. It is still used to test results.

### Usage

```
get_pdp_predictions_seq(df, imp, m, degree = 4, bins = 5, .f_predict = predict)
```

### Arguments

<code>df</code>	dataframe, training data
<code>imp</code>	dataframe, with not more then two columns one of them numeric containing importance measures and one character or factor column containing corresponding variable names as found in training data.
<code>m</code>	model object
<code>degree</code>	integer, number of top important variables to select. For plotting more than 4 will result in two many flows and the alluvial plot will not be very readable, Default: 4
<code>bins</code>	integer, number of bins for numeric variables, increasing this number might result in too many flows, Default: 5
<code>.f_predict</code>	corresponding model <code>predict()</code> function. Needs to accept 'm' as the first parameter and use the 'newdata' parameter. Supply a wrapper for predict functions with x-y syntax. For parallel processing the predict method of object classes will not always get imported correctly to the worker environment. We can pass the correct predict method via this parameter for example <code>randomForest::predict.randomForest</code> . Note that a lot of modeling packages do not export the predict method explicitly and it can only be found using <code>:::</code> .

### See Also

[get\\_pdp\\_predictions](#)

---

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,
  round_numeric = T,
  digits = 2,
  NA_label = "NA"
)
```

### Arguments

x	dataframe with numeric variables, or numeric vector
bins	number of bins for numerical variables, passed to cut as breaks parameter, Default: 5
bin_labels	labels for the bins from low to high, Default: c("LL", "ML", "M", "MH", "HH"). Can also be one of c('mean', 'median', 'min_max', 'cuts'), the corresponding summary function will supply the labels.
center	logical, Default: T
scale	logical, Default: T
transform	logical, apply Yeo Johnson Transformation, Default: T
round_numeric	logical, rounds numeric results if bin_labels is supplied with a supported summary function name.
digits	integer, number of digits to round to
NA_label	character vector, define label for missing data, Default: 'NA'

### Value

dataframe

## Examples

```
summary( mtcars2 )
summary( manip_bin_numerics(mtcars2) )
summary( manip_bin_numerics(mtcars2, bin_labels = 'mean'))
summary( manip_bin_numerics(mtcars2, bin_labels = 'cuts'
, scale = FALSE, center = FALSE, transform = FALSE))
```

---

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 )
# does not work for decimal numbers
manip_factor_2_numeric(factor(c("A12", "B55", "10e4")))
manip_factor_2_numeric(factor(c("1.56", "4.56", "8.4")))
```

---

`manip_get_ggplot_data` *Get ggplot data*

---

**Description**

retrieve ggplot 2 plot data from S3 and S7 (>= ggplot2 v4.0)

**Usage**

`manip_get_ggplot_data(p)`

**Arguments**

`p` ggplot object

---

`mtcars2` *mtcars dataset with cyl, vs, am ,gear, carb as factor variables and car model names as id*

---

**Description**

mtcars dataset with cyl, vs, am ,gear, carb as factor variables and car model names as id

**Usage**

`mtcars2`

**Format**

A data frame with 32 rows and 12 variables

**mpg** Miles/(US) gallon

**cyl** Number of cylinders

**disp** Displacement (cu.in.)

**hp** Gross horsepower

**drat** Rear axle ratio

**wt** Weight (1000 lbs)

**qsec** 1/4 mile time

**vs** Engine

**am** Transmission

**gear** Number of forward gears

**carb** Number of carburetors

**ids** car model name

**Source**

datasets

---

 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**

```
require(magrittr)

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

## Not run:
# more examples-----

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**

uses `c('#FF0065', '#009850', '#A56F2B', '#005EAA', '#710500', '#7B5380', '#9DD1D1')` and then adds all unique values found in all qualitative RColorBrewer palettes

**Usage**

```
palette_qualitative()
```

**Value**

vector with hex values

**See Also**

[RColorBrewer](#)

**Examples**

```
palette_qualitative()
```

---

plot_all_hists	<i>plot marginal histograms of alluvial plot</i>
----------------	--

---

**Description**

will create gtable with density histograms and frequency plots of all variables of a given alluvial plot.

**Usage**

```
plot_all_hists(p, data_input, top = TRUE, keep_labels = FALSE, ...)
```

**Arguments**

p	alluvial plot
data_input	dataframe, input data that was used to create dataframe
top	logical, position of histograms, if FALSE adds them at the bottom, Default: TRUE
keep_labels	logical, keep title and caption, Default: FALSE
...	additional arguments for specific alluvial plot types: pred_train can be used to pass training predictions for model response alluvials

**Value**

gtable

**See Also**

[arrangeGrob](#)

[add\\_marginal\\_histograms](#)

## Examples

```
## Not run:  
p = alluvial_wide(mtcars2, max_variables = 3)  
plot_all_hists(p, mtcars2)  
  
## End(Not run)
```

---

plot\_condensation      *Plot dataframe condensation potential*

---

## 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

```
plot_condensation(mtcars2)  
  
plot_condensation(mtcars2, first = 'disp')
```

---

plot_hist	<i>plot histogram of alluvial plot variable</i>
-----------	---

---

**Description**

helper function used by add\_marginal\_histograms

**Usage**

```
plot_hist(var, p, data_input, ...)
```

**Arguments**

var	character vector, variable name
p	alluvial plot
data_input	dataframe used to create alluvial plot
...	additional arguments for specific alluvial plot types: pred_train can be used to pass training predictions for model response alluvials

**Value**

ggplot object

---

plot_imp	<i>plot feature importance</i>
----------	--------------------------------

---

**Description**

plot important features of model response alluvial as bars

**Usage**

```
plot_imp(p, data_input, truncate_at = 50, color = "darkgrey")
```

**Arguments**

p	alluvial plot
data_input	dataframe used to generate alluvial plot
truncate_at	integer, limit number of features to that value, Default: 50
color	character vector, Default: 'darkgrey'

**Value**

ggplot object

**Examples**

```
## Not run:
df = mtcars2[, ! names(mtcars2) %in% 'ids' ]

train = caret::train( disp ~ .
                      , df
                      , method = 'rf'
                      , trControl = caret::trainControl( method = 'none' )
                      , importance = TRUE )

pred_train = caret::predict.train(train, df)

p = alluvial_model_response_caret(train, degree = 3, pred_train = pred_train)

plot_imp(p, mtcars2)

## End(Not run)
```

---

quarterly\_flights

*Quarterly mean arrival delay times for a set of 402 flights*


---

**Description**

Created from nycflights13::flights

**Usage**

```
quarterly_flights
```

**Format**

A data frame with 1608 rows and 6 variables

**tailnum** a unique identifier created from tailnum, origin, destination and carrier

**carrier** carrier code

**origin** origin code

**dest** destination code

**qu** quarter

**mean\_arr\_delay** average delay on arrival as either on\_time or late

**Source**

nycflights13::flights

---

quarterly_sunspots	<i>Quarterly mean relative sunspots number from 1749-1983</i>
--------------------	---

---

**Description**

Quarterly mean relative sunspots number from 1749-1983

**Usage**

```
quarterly_sunspots
```

**Format**

A data frame with 940 rows and 4 variables

**year**

**qu** quarter

**spots** total number of sunspots

**mean\_spots\_per\_year**

**Source**

Andrews, D. F. and Herzberg, A. M. (1985) Data: A Collection of Problems from Many Fields for the Student and Research Worker. New York: Springer-Verlag.

---

tidy_imp	<i>tidy up dataframe containing model feature importance</i>
----------	--

---

**Description**

returns dataframe with exactly two columns, vars and imp and aggregates dummy encoded variables. Helper function called by all functions that take an imp parameter. Can be called manually if formula for aggregating dummy encoded variables must be modified.

**Usage**

```
tidy_imp(imp, df, .f = max, resp_var = NULL)
```

**Arguments**

**imp** dataframe or matrix with feature importance information

**df** dataframe, modeling training data

**.f** window function, Default: max

**resp\_var** character, prediction variable, can usually be inferred from imp and df. It does not work for all models and needs to be specified in those cases.

**Value**

dataframe

**vars** character column with feature names

**imp** numerical column, importance values

**Examples**

```
# randomforest
df = mtcars2[, ! names(mtcars2) %in% 'ids' ]
m = randomForest::randomForest( disp ~ ., df)
imp = m$importance
tidy_imp(imp, df)
```

---

titanic

*titanic data set'*

---

**Description**

titanic data set'

**Usage**

titanic

**Format**

A data frame with 891 rows and 10 variables

**Survived** Survived

**Pclass** Pclass

**Sex** Sex

**Age** Age

**SibSp** SibSp

**Parch** Parch

**Fare** Fare

**Cabin** Cabin

**Embarked** Embarked

**title** title

**Source**

datasets

# Index

## \* datasets

- mtcars2, [24](#)
  - quarterly\_flights, [32](#)
  - quarterly\_sunspots, [33](#)
  - titanic, [34](#)
- add\_imp\_plot, [3](#)
- add\_marginal\_histograms, [4](#), [29](#)
- alluvial\_long, [5](#)
- alluvial\_model\_response, [8](#), [10](#), [12](#)
- alluvial\_model\_response\_caret, [9](#), [10](#)
- alluvial\_model\_response\_parsnip, [12](#)
- alluvial\_wide, [7](#), [9](#), [11](#), [13](#), [14](#), [14](#), [16](#), [18](#)
- arrangeGrob, [3](#), [4](#), [29](#)
- check\_pkg\_installed, [17](#)
- extractPrediction, [11](#), [14](#)
- geom\_flow, [7](#), [16](#)
- geom\_stratum, [7](#), [16](#)
- get\_data\_space, [8–12](#), [14](#), [17](#)
- get\_pdp\_predictions, [8](#), [9](#), [11](#), [14](#), [19](#), [21](#)
- get\_pdp\_predictions\_seq, [21](#)
- manip\_bin\_numerics, [6](#), [7](#), [9](#), [11](#), [13](#), [16](#), [18](#),  
[22](#)
- manip\_factor\_2\_numeric, [23](#)
- manip\_get\_ggplot\_data, [24](#)
- mtcars2, [24](#)
- palette\_filter, [25](#)
- palette\_increase\_length, [26](#)
- palette\_plot\_intensity, [27](#), [28](#)
- palette\_plot\_rgp, [27](#), [28](#)
- palette\_qualitative, [28](#)
- plan, [11](#), [14](#), [20](#)
- plot\_all\_hists, [29](#)
- plot\_condensation, [30](#)
- plot\_hist, [31](#)
- plot\_imp, [3](#), [31](#)
- quarterly\_flights, [32](#)
- quarterly\_sunspots, [33](#)
- quosure, [30](#)
- RColorBrewer, [29](#), [30](#)
- reexports, [30](#)
- str\_detect, [23](#)
- tidy\_imp, [33](#)
- titanic, [34](#)
- varImp, [11](#), [14](#)