Package ‘healthyR.ai’

April 3, 2023

Title The Machine Learning and AI Modeling Companion to 'healthyR'

Version 0.0.13

Description Hospital machine learning and ai data analysis workflow tools, modeling, and automations. This library provides many useful tools to review common administrative hospital data. Some of these include predicting length of stay, and readmits. The aim is to provide a simple and consistent verb framework that takes the guesswork out of everything.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

URL https://github.com/spsanderson/healthyR.ai

BugReports https://github.com/spsanderson/healthyR.ai/issues

Imports magrittr, rlang (>= 0.1.2), yardstick (>= 0.0.8), utils,
   broom, ggrepel, tibble, dplyr, ggplot2, tidyr, forcats, recipes
   (>= 1.0.0), purrr, h2o, stats, dials, parsnip, tune, workflows,
   modeltime

Suggests rmarkdown, knitr, roxygen2, healthyR.data, scales,
   tidyselect, janitor, timetk, plotly, rsample, kknn, hardhat,
   uwot, stringr

VignetteBuilder knitr

Depends R (>= 3.3)

NeedsCompilation no

Author Steven Sanderson [aut, cre, cph]

Maintainer Steven Sanderson <spsanderson@gmail.com>

Repository CRAN

Date/Publication 2023-04-03 00:20:02 UTC
# R topics documented:

- `color_blind` ....................................................... 3
- `get_juiced_data` ................................................. 4
- `hai_auto_c50` ..................................................... 5
- `hai_auto_cubist` .................................................. 7
- `hai_auto_earth` ................................................... 9
- `hai_auto_glmnet` ............................................... 10
- `hai_auto_knn` .................................................... 12
- `hai_auto_ranger` ............................................... 14
- `hai_auto_svm_poly` ............................................. 15
- `hai_auto_svm_rbf` .............................................. 17
- `hai_auto_wfw_metrics` ......................................... 19
- `hai_auto_xgboost` .............................................. 20
- `hai_c50_data_prepper` ......................................... 22
- `hai_control_chart` .............................................. 23
- `hai_cubist_data_prepper` ...................................... 24
- `hai_data_impute` ................................................. 26
- `hai_data_poly` ................................................... 28
- `hai_data_scale` ................................................... 30
- `hai_data_transform` ............................................. 32
- `hai_data_trig` ..................................................... 35
- `hai_default_classification_metric_set` ....................... 37
- `hai_default_regression_metric_set` .......................... 37
- `hai_density_hist_plot` ......................................... 38
- `hai_density_plot` ............................................... 39
- `hai_density_qq_plot` ........................................... 41
- `hai_distribution_comparison_tbl` .............................. 42
- `hai_earth_data_prepper` ....................................... 44
- `hai_fourier_augment` ........................................... 45
- `hai_fourier_discrete_augment` ................................. 47
- `hai_fourier_discrete_vec` ..................................... 48
- `hai_fourier_vec` .................................................. 50
- `hai_get_density_data_tbl` ...................................... 52
- `hai_get_dist_data_tbl` ......................................... 53
- `hai_glmnet_data_prepper` ...................................... 54
- `hai_histogram_facet_plot` ..................................... 55
- `hai_hyperbolic_augment` ....................................... 56
- `hai_hyperbolic_vec` ............................................. 58
- `hai_kmeans_automl` ............................................. 59
- `hai_kmeans_automl_predict` ................................... 61
- `hai_kmeans_mapped_tbl` ....................................... 62
- `hai_kmeans_obj` .................................................. 63
- `hai_kmeans_scree_data_tbl` ................................... 65
- `hai_kmeans_scree_plt` ......................................... 66
- `hai_kmeans_tidy_tbl` ........................................... 67
- `hai_kmeans_user_item_tbl` .................................... 69
- `hai_knn_data_prepper` ......................................... 70
color_blind

Provide Colorblind Compliant Colors

Description

8 Hex RGB color definitions suitable for charts for colorblind people.

Usage

color_blind()

Details

This function is used in others in order to help render plots for those that are color blind.
Value
A vector of 8 Hex RGB definitions.

Author(s)
Steven P. Sanderson II, MPH

See Also
Other Color_Blind: hai_scale_color_colorblind(), hai_scale_fill_colorblind()

Examples
color_blind()

description
This is a simple function that will get the juiced data from a recipe.

Usage
get_juiced_data(.recipe_object)

Arguments
.recipe_object The recipe object you want to pass.

Details
Instead of typing out something like: recipe_object %>% prep() %>% juice() %>% glimpse()

Value
A tibble of the prepped and juiced data from the given recipe

Author(s)
Steven P. Sanderson II, MPH

See Also
Other Data Wrangling: pca_your_recipe()
Examples

```r
evaluate {
  suppressPackageStartupMessages(library(timetk))
  suppressPackageStartupMessages(library(dplyr))
  suppressPackageStartupMessages(library(purrr))
  suppressPackageStartupMessages(library(healthyR.data))
  suppressPackageStartupMessages(library(rsample))
  suppressPackageStartupMessages(library(recipes))

  data_tbl <- healthyR_data %>%
    select(visit_end_date_time) %>%
    summarise_by_time(
      .date_var = visit_end_date_time,
      .by = "month",
      value = n()
    ) %>%
    set_names("date_col", "value") %>%
    filter_by_time(
      .date_var = date_col,
      .start_date = "2013",
      .end_date = "2020"
    )

  splits <- initial_split(data = data_tbl, prop = 0.8)

  rec_obj <- recipe(value ~ ., training(splits))

  get_juiced_data(rec_obj)
}
```

**Boilerplate Workflow**

**Description**

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)

**Usage**

```r
hai_auto_c50(
  .data,
  .rec_obj,
  .splits_obj = NULL,
  .rsamp_obj = NULL,
)```
.tune = TRUE,
.grid_size = 10,
.num_cores = 1,
.best_metric = "f_meas",
.model_type = "classification"
)

Arguments

.data The data being passed to the function. The time-series object.
.rec_obj This is the recipe object you want to use. You can use hai_c50_data_prepper() an automatic recipe_object.
.splits_obj NULL is the default, when NULL then one will be created.
.rsamp_obj NULL is the default, when NULL then one will be created. It will default to creating an rsample::mc_cv() object.
.tune Default is TRUE, this will create a tuning grid and tuned workflow
.grid_size Default is 10
.num_cores Default is 1
.best_metric Default is "f_meas". You can choose a metric depending on the model_type used. If regression then see hai_default_regression_metric_set(), if classification then see hai_default_classification_metric_set().
.model_type Default is classification, can also be regression.

Details

This uses the parsnip::boost_tree() with the engine set to C5.0

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Boiler_Plate: hai_auto_cubist(), hai_auto_earth(), hai_auto_glmnet(), hai_auto_knn(), hai_auto_ranger(), hai_auto_svm_poly(), hai_auto_svm_rbf(), hai_auto_wflw_metrics(), hai_auto_xgboost()

Other C5.0: hai_c50_data_prepper()
Examples

```r
## Not run:
data <- iris

rec_obj <- hai_c50_data_prepper(data, Species ~ .)

auto_c50 <- hai_auto_c50(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas",
  .model_type = "classification"
)

auto_c50$recipe_info

## End(Not run)
```

---

**Boilerplate Workflow**

**Description**

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)

**Usage**

```r
hai_auto_cubist(  
  .data,  
  .rec_obj,  
  .splits_obj = NULL,  
  .rsamp_obj = NULL,  
  .tune = TRUE,  
  .grid_size = 10,  
  .num_cores = 1,  
  .best_metric = "rmse"
)
```
Arguments

- `.data` The data being passed to the function. The time-series object.
- `.rec_obj` This is the recipe object you want to use. You can use `hai_cubist_data_prepper()` an automatic recipe object.
- `.splits_obj` NULL is the default, when NULL then one will be created.
- `.rsamp_obj` NULL is the default, when NULL then one will be created. It will default to creating an `rsample::mc_cv()` object.
- `.tune` Default is TRUE, this will create a tuning grid and tuned workflow
- `.grid_size` Default is 10
- `.num_cores` Default is 1
- `.best_metric` Default is "rmse". The only .model_type you can use with Cubist is regression so use `hai_default_regression_metric_set()` to get the available metrics. Because of this the .model_type parameter is omitted from this function.

Details

This uses the `parsnip::cubist_rules()` with the engine set to cubist

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Boiler_Plate: `hai_auto_c50()`, `hai_auto_earth()`, `hai_auto_glmnet()`, `hai_auto_knn()`, `hai_auto_ranger()`, `hai_auto_svm_poly()`, `hai_auto_svm_rbf()`, `hai_auto_wflw_metrics()`, `hai_auto_xgboost()`

Other cubist: `hai_cubist_data_prepper()`

Examples

```r
## Not run:
data <- mtcars
rec_obj <- hai_cubist_data_prepper(data, mpg ~ .)

auto_cube <- hai_auto_cubist(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "rmse"
)

auto_cube$recipe_info
```
### Boilerplate Workflow

**Description**

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)

**Usage**

```r
hai_auto_earth(
  .data,
  .rec_obj,
  .splits_obj = NULL,
  .rsamp_obj = NULL,
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .best_metric = "f_meas",
  .model_type = "classification"
)
```

**Arguments**

- **.data** The data being passed to the function. The time-series object.
- **.rec_obj** This is the recipe object you want to use. You can use `hai_earth_data_prepper()` to create an automatic recipe object.
- **.splits_obj** NULL is the default, when NULL then one will be created.
- **.rsamp_obj** NULL is the default, when NULL then one will be created. It will default to creating an `rsample::mc_cv()` object.
- **.tune** Default is TRUE, this will create a tuning grid and tuned workflow
- **.grid_size** Default is 10
- **.num_cores** Default is 1
- **.best_metric** Default is "f_meas". You can choose a metric depending on the model_type used. If regression then see `hai_default_regression_metric_set()`, if classification then see `hai_default_classification_metric_set()`.
- **.model_type** Default is classification, can also be regression.
Details

This uses the `parsnip::mars()` with the engine set to `earth`

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

http://uc-r.github.io/mars

Other Boiler_Plate: `hai_auto_c50()`, `hai_auto_cubist()`, `hai_auto_glmnet()`, `hai_auto_knn()`, `hai_auto_ranger()`, `hai_auto_svm_poly()`, `hai_auto_svm_rbf()`, `hai_auto_wflw_metrics()`, `hai_auto_xgboost()`

Other Earth: `hai_earth_data_prepper()`

Examples

```r
## Not run:
data <- iris

rec_obj <- hai_earth_data_prepper(data, Species ~ .)

auto_earth <- hai_auto_earth(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas",
  .model_type = "classification"
)

auto_earth$recipe_info

## End(Not run)
```

---

**Description**

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)
**hai_auto_glmnet**

**Usage**

```r
hai_auto_glmnet(
  .data,
  .rec_obj,
  .splits_obj = NULL,
  .rsamp_obj = NULL,
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .best_metric = "f_meas",
  .model_type = "classification"
)
```

**Arguments**

- **.data**: The data being passed to the function. The time-series object.
- **.rec_obj**: This is the recipe object you want to use. You can use `hai_glmnet_data_prepper()` with an automatic recipe object.
- **.splits_obj**: `NULL` is the default, when `NULL` then one will be created.
- **.rsamp_obj**: `NULL` is the default, when `NULL` then one will be created. It will default to creating an `rsample::mc_cv()` object.
- **.tune**: Default is `TRUE`, this will create a tuning grid and tuned workflow.
- **.grid_size**: Default is 10
- **.num_cores**: Default is 1
- **.best_metric**: Default is "f_meas". You can choose a metric depending on the model_type used. If regression then see `hai_default_regression_metric_set()`, if classification then see `hai_default_classification_metric_set()`.
- **.model_type**: Default is `classification`, can also be regression.

**Details**

This uses the `parsnip::multinom_reg()` with the engine set to `glmnet`.

**Value**

A list

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Boiler_Plate: `hai_auto_c50()`, `hai_auto_cubist()`, `hai_auto_earth()`, `hai_auto_knn()`, `hai_auto_ranger()`, `hai_auto_svm_poly()`, `hai_auto_svm_rbf()`, `hai_auto_wflw_metrics()`, `hai_auto_xgboost()`
Examples

```r
## Not run:
data <- iris

rec_obj <- hai_glmnet_data_prepper(data, Species ~ .)

auto_glm <- hai_auto_glmnet(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas",
  .model_type = "classification"
)

auto_glm$recipe_info
```

## End(Not run)

---

**hai_auto_knn**

**Boilerplate Workflow**

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)

Usage

```r
hai_auto_knn(
  .data,  
  .rec_obj,  
  .splits_obj = NULL,  
  .rsamp_obj = NULL,  
  .tune = TRUE,  
  .grid_size = 10,  
  .num_cores = 1,  
  .best_metric = "rmse",  
  .model_type = "regression"
)
```
Arguments

.data The data being passed to the function. The time-series object.
.rec_obj This is the recipe object you want to use. You can use \texttt{hai_knn_data_prepper()} an automatic recipe_object.
.splits_obj NULL is the default, when NULL then one will be created.
.rsamp_obj NULL is the default, when NULL then one will be created. It will default to creating an \texttt{rsample::mc_cv()} object.
.tune Default is TRUE, this will create a tuning grid and tuned workflow
.grid_size Default is 10
.num_cores Default is 1
.best_metric Default is "rmse". You can choose a metric depending on the model_type used. If regression then see \texttt{hai_default_regression_metric_set()}. if classification then see \texttt{hai_default_classification_metric_set()}
.model_type Default is regression, can also be classification.

Details

This uses the \texttt{parsnip::nearest_neighbor()} with the engine set to \texttt{kknn}

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Boiler_Plate: \texttt{hai_auto_c50()}, \texttt{hai_auto_cubist()}, \texttt{hai_auto_earth()}, \texttt{hai_auto_glmnet()}, \texttt{hai_auto_ranger()}, \texttt{hai_auto_svm_poly()}, \texttt{hai_auto_svm_rbf()}, \texttt{hai_auto_wflw_metrics()}, \texttt{hai_auto_xgboost()}

Examples

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

data <- iris

rec_obj <- hai_knn_data_prepper(data, Species ~ .)

auto_knn <- hai_auto_knn(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas",
  .model_type = "classification"
)
```
auto_knn$recipe_info

## End(Not run)

hai_auto_ranger  

**Boilerplate Workflow**

**Description**

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)

**Usage**

```r
hai_auto_ranger(
  .data,
  .rec_obj,
  .splits_obj = NULL,
  .rsamp_obj = NULL,
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .best_metric = "f_meas",
  .model_type = "classification"
)
```

**Arguments**

- `.data` The data being passed to the function. The time-series object.
- `.rec_obj` This is the recipe object you want to use. You can use `hai_ranger_data_prepper()` an automatic recipe object.
- `.splits_obj` NULL is the default, when NULL then one will be created.
- `.rsamp_obj` NULL is the default, when NULL then one will be created. It will default to creating an `rsample::mc_cv()` object.
- `.tune` Default is TRUE, this will create a tuning grid and tuned workflow
- `.grid_size` Default is 10
- `.num_cores` Default is 1
- `.best_metric` Default is "f_meas". You can choose a metric depending on the model_type used. If regression then see `hai_default_regression_metric_set()`, if classification then see `hai_default_classification_metric_set()`.
- `.model_type` Default is classification, can also be regression.


### Details
This uses the `parsnip::rand_forest()` with the engine set to `kernlab`

### Value
A list

### Author(s)
Steven P. Sanderson II, MPH

### See Also
- [https://parsnip.tidymodels.org/reference/rand_forest.html](https://parsnip.tidymodels.org/reference/rand_forest.html)
- Other Boiler Plate: `hai_auto_c50()`, `hai_auto_cubist()`, `hai_auto_earth()`, `hai_auto_glmnet()`, `hai_auto_knn()`, `hai_auto_svm_poly()`, `hai_auto_svm_rbf()`, `hai_auto_wflw_metrics()`, `hai_auto_xgboost()`
- Other Ranger: `hai_ranger_data_prepper()`

### Examples
```r
## Not run:
data <- iris
rec_obj <- hai_ranger_data_prepper(data, Species ~ .)

auto_ranger <- hai_auto_ranger(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas"
)

auto_ranger$recipe_info

## End(Not run)
```

### Description
This is a boilerplate function to create automatically the following:
- recipe
- model specification
- workflow
- tuned model (grid ect)
Usage

hai_auto_svm_poly(
  .data,
  .rec_obj,
  .splits_obj = NULL,
  .rsamp_obj = NULL,
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .best_metric = "f_meas",
  .model_type = "classification"
)

Arguments

.data The data being passed to the function. The time-series object.
.rec_obj This is the recipe object you want to use. You can use hai_svm_poly_data_prepper() an automatic recipe_object.
.splits_obj NULL is the default, when NULL then one will be created.
.rsamp_obj NULL is the default, when NULL then one will be created. It will default to creating an rsample::mc_cv() object.
.tune Default is TRUE, this will create a tuning grid and tuned workflow
.grid_size Default is 10
.num_cores Default is 1
.best_metric Default is "f_meas". You can choose a metric depending on the model_type used. If regression then see hai_default_regression_metric_set(). if classification then see hai_default_classification_metric_set().
.model_type Default is classification, can also be regression.

Details

This uses the parsnip::svm_poly() with the engine set to kernlab

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/svm_poly.html

Other Boiler_Plate: hai_auto_c50(), hai_auto_cubist(), hai_auto_earth(), hai_auto_glmnet(),
    hai_auto_knn(), hai_auto_ranger(), hai_auto_svm_rbf(), hai_auto_wflw_metrics(), hai_auto_xgboost()

Other SVM_Poly: hai_svm_poly_data_prepper()
Examples

## Not run:
data <- iris

rec_obj <- hai_svm_poly_data_prepper(data, Species ~ .)

auto_svm_poly <- hai_auto_svm_poly(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas"
)

auto_svm_poly$recipe_info

## End(Not run)

---

hai_auto_svm_rbf  

### Boilerplate Workflow

**Description**

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid etc)

**Usage**

```r
hai_auto_svm_rbf(
  .data,  
  .rec_obj,  
  .splits_obj = NULL,  
  .rsamp_obj = NULL,  
  .tune = TRUE,  
  .grid_size = 10,  
  .num_cores = 1,  
  .best_metric = "f_meas",  
  .model_type = "classification"
)
```
Arguments

.data   The data being passed to the function. The time-series object.
.rec_obj This is the recipe object you want to use. You can use hai_svm_rbf_data_prepper() an automatic recipe_object.
.splits_obj NULL is the default, when NULL then one will be created.
.rsamp_obj NULL is the default, when NULL then one will be created. It will default to creating an rsample::mc_cv() object.
.tune   Default is TRUE, this will create a tuning grid and tuned workflow
.grid_size Default is 10
.num_cores Default is 1
.best_metric Default is "f_meas". You can choose a metric depending on the model_type used. If regression then see hai_default_regression_metric_set(), if classification then see hai_default_classification_metric_set().
.model_type Default is classification, can also be regression.

Details

This uses the parsnip::svm_rbf() with the engine set to kernlab

Value

A list

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/svm_rbf.html

Other Boiler_Plate: hai_auto_c50(), hai_auto_cubist(), hai_auto_earth(), hai_auto_glmnet(), hai_auto_knn(), hai_auto_ranger(), hai_auto_svm_poly(), hai_auto_wflw_metrics(), hai_auto_xgboost()

Other SVM_RBF: hai_svm_rbf_data_prepper()

Examples

## Not run:
data <- iris

rec_obj <- hai_svm_rbf_data_prepper(data, Species ~ .)

auto_rbf <- hai_auto_svm_rbf(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas"
)
Collect Metrics from Boilerplate Workflows

Description
This function will extract the metrics from the hai_auto_ boilerplate functions.

Usage
hai_auto_wflw_metrics(.data)

Arguments
.data The output of the hai_auto_ boilerplate function in it’s entirety.

Details
This function will extract the metrics from the hai_auto_ boilerplate functions. This function looks for a specific attribute from the hai_auto_ functions so that it will extract the tuned_results from the tuning process if it has indeed been tuned.

Value
A tibble

Author(s)
Steven P. Sanderson II, MPH

See Also
Other Boiler_Plate: hai_auto_c50(), hai_auto_cubist(), hai_auto_earth(), hai_auto_glmnet(), hai_auto_knn(), hai_auto_ranger(), hai_auto_svm_poly(), hai_auto_svm_rbf(), hai_auto_xgboost()

Examples
## Not run:
data <- iris
rec_obj <- hai_knn_data_prepper(data, Species ~ .)

auto_knn <- hai_auto_knn(
  .data = data,
  .rec_obj = rec_obj,
hai_auto_xgboost

Boilerplate Workflow

Description

This is a boilerplate function to create automatically the following:

- recipe
- model specification
- workflow
- tuned model (grid ect)

Usage

hai_auto_xgboost(
  .data,
  .rec_obj,
  .splits_obj = NULL,
  .rsamp_obj = NULL,
  .tune = TRUE,
  .grid_size = 10,
  .num_cores = 1,
  .best_metric = "f_meas",
  .model_type = "classification"
)

Arguments

.data The data being passed to the function. The time-series object.
.rec_obj This is the recipe object you want to use. You can use hai_xgboost_data_prepper() an automatic recipe_object.
.splits_obj NULL is the default, when NULL then one will be created.
.rsamp_obj NULL is the default, when NULL then one will be created. It will default to creating an rsample::mc_cv() object.
.tune Default is TRUE, this will create a tuning grid and tuned workflow
**Details**

This uses the `parsnip::boost_tree()` with the engine set to xgboost.

**Value**

A list

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**


Other Boiler_Plate: `hai_auto_c50()`, `hai_auto_cubist()`, `hai_auto_earth()`, `hai_auto_glmnet()`, `hai_auto_knn()`, `hai_auto_ranger()`, `hai_auto_svm_poly()`, `hai_auto_svm_rbf()`, `hai_auto_wflw_metrics()`

**Examples**

```r
## Not run:
data <- iris
rec_obj <- hai_xgboost_data_prepper(data, Species ~ .)

auto_xgb <- hai_auto_xgboost(
  .data = data,
  .rec_obj = rec_obj,
  .best_metric = "f_meas"
)

auto_xgb$recipe_info

## End(Not run)
```
**hai_c50_data_prepper**  
*Prep Data for C5.0 - Recipe*

**Description**
Automatically prep a data.frame/tibble for use in the C5.0 algorithm.

**Usage**

```r
hai_c50_data_prepper(.data, .recipe_formula)
```

**Arguments**
- `.data`  
The data that you are passing to the function. Can be any type of data that is accepted by the `data` parameter of the `recipes::recipe()` function.
- `.recipe_formula`  
The formula that is going to be passed. For example if you are using the `iris` data then the formula would most likely be something like `Species ~ .`.

**Details**
This function will automatically prep your data.frame/tibble for use in the C5.0 algorithm. The C5.0 algorithm is a lazy learning classification algorithm. It expects data to be presented in a certain fashion.

This function will output a recipe specification.

**Value**
A recipe object

**Author(s)**
Steven P. Sanderson II, MPH

**See Also**
- [https://www.rulequest.com/see5-unix.html](https://www.rulequest.com/see5-unix.html)
- Other Preprocessor: `hai_cubist_data_prepper()`, `hai_data_impute()`, `hai_data_poly()`, `hai_data_scale()`, `hai_data_transform()`, `hai_data_trig()`, `hai_earth_data_prepper()`, `hai_glmnet_data_prepper()`, `hai_knn_data_prepper()`, `hai_ranger_data_prepper()`, `hai_svm_poly_data_prepper()`, `hai_svm_rbf_data_prepper()`, `hai_xgboost_data_prepper()`
- Other C5.0: `hai_auto_c50()`
### hai_control_chart

Create a control chart, aka Shewhart chart: https://en.wikipedia.org/wiki/Control_chart.

#### Description

Create a control chart, aka Shewhart chart. 

#### Usage

```r
hai_control_chart(
  .data,
  .value_col,
  .x_col,
  .center_line = mean,
  .std_dev = 3,
  .plt_title = NULL,
  .plt_caption = NULL,
  .plt_font_size = 11,
  .print_plot = TRUE
)
```

#### Arguments

- `data` data frame or a path to a csv file that will be read in
- `value_col` variable of interest mapped to y-axis (quoted, ie as a string)
- `x_col` variable to go on the x-axis, often a time variable. If unspecified row indices will be used (quoted)
- `center_line` Function used to calculate central tendency. Defaults to mean
- `std_dev` Number of standard deviations above and below the central tendency to call a point influenced by "special cause variation." Defaults to 3
- `plt_title` Plot title
- `plt_caption` Plot caption
- `plt_font_size` Font size; text elements will be scaled to this
- `print_plot` Print the plot? Default = TRUE. Set to FALSE if you want to assign the plot to a variable for further modification, as in the last example.

### Examples

```r
library(ggplot2)

hai_c50_data_prepper(.data = Titanic, .recipe_formula = Survived ~ .)
rec_obj <- hai_c50_data_prepper(Titanic, Survived ~ .)
get_juiced_data(rec_obj)
```
Details

Control charts, also known as Shewhart charts (after Walter A. Shewhart) or process-behavior charts, are a statistical process control tool used to determine if a manufacturing or business process is in a state of control. It is more appropriate to say that the control charts are the graphical device for Statistical Process Monitoring (SPM). Traditional control charts are mostly designed to monitor process parameters when underlying form of the process distributions are known. However, more advanced techniques are available in the 21st century where incoming data streaming can be monitored even without any knowledge of the underlying process distributions. Distribution-free control charts are becoming increasingly popular.

Value

Generally called for the side effect of printing the control chart. Invisibly, returns a ggplot object for further customization.

Author(s)

Steven P. Sanderson II, MPH

Examples

```r
data_tbl <- tibble::tibble(
  day = sample(c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday"), 100, TRUE),
  person = sample(c("Tom", "Jane", "Alex"), 100, TRUE),
  count = rbinom(100, 20, ifelse(day == "Friday", .5, .2)),
  date = Sys.Date() - sample.int(100)
)

hai_control_chart(.data = data_tbl, .value_col = count, .x_col = date)
```

# In addition to printing or writing the plot to file, hai_control_chart returns the plot as a ggplot2 object, which you can then further customize

```r
library(ggplot2)
my_chart <- hai_control_chart(data_tbl, count, date)
my_chart +
  ylab("Number of Adverse Events") +
  scale_x_date(name = "Week of ... ", date_breaks = "week") +
  theme(axis.text.x = element_text(angle = -90, vjust = 0.5, hjust = 1))
```
Description
Automatically prep a data.frame/tibble for use in the cubist algorithm.

Usage
hai_cubist_data_prepper(.data, .recipe_formula)

Arguments
.data The data that you are passing to the function. Can be any type of data that is accepted by the data parameter of the recipes::recipe() function.
.recipe_formula The formula that is going to be passed. For example if you are using the diamonds data then the formula would most likely be something like price ~ .

Details
This function will automatically prep your data.frame/tibble for use in the cubist algorithm. The cubist algorithm is for regression only.
This function will output a recipe specification.

Value
A recipe object

Author(s)
Steven P. Sanderson II, MPH

See Also
https://rulequest.com/cubist-info.html
Other Preprocessor: hai_c50_data_prepper(), hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig(), hai_earth_data_prepper(), hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_poly_data_prepper(), hai_svm_rbf_data_prepper(), hai_xgboost_data_prepper()
Other cubist: hai_auto_cubist()

Examples
library(ggplot2)
hai_cubist_data_prepper(.data = diamonds, .recipe_formula = price ~ .)
rec_obj <- hai_cubist_data_prepper(diamonds, price ~ .)
get_juiced_data(rec_obj)


**hai_data_impute**

*Data Preprocessor - Imputation*

**Description**

Takes in a recipe and will impute missing values using a selected recipe. To call the recipe use a quoted argument like "median" or "bagged".

**Usage**

```r
hai_data_impute(
  .recipe_object = NULL,
  ...,  
  .seed_value = 123,
  .type_of_imputation = "mean",
  .number_of_trees = 25,
  .neighbors = 5,
  .mean_trim = 0,
  .roll_statistic,
  .roll_window = 5
)
```

**Arguments**

- **.recipe_object** The data that you want to process
- **...** One or more selector functions to choose variables to be imputed. When used with imp_vars, these dots indicate which variables are used to predict the missing data in each variable. See selections() for more details
- **.seed_value** To make results reproducible, set the seed.
- **.type_of_imputation** This is a quoted argument and can be one of the following:
  - "bagged"
  - "knn"
  - "linear"
  - "lower"
  - "mean"
  - "median"
  - "mode"
  - "roll"
- **.number_of_trees** This is used for the `recipes::step_impute_bag()` trees parameter. This should be an integer.
- **.neighbors** This should be filled in with an integer value if `.type_of_imputation` selected is "knn".
Details

This function will get your data ready for processing with many types of ml/ai models.

This is intended to be used inside of the data processor and therefore is an internal function. This documentation exists to explain the process and help the user understand the parameters that can be set in the pre-processor function.

Value

A list object

Author(s)

Steven P. Sanderson II, MPH

See Also

step_impute_bag
recipes::step_impute_bag()
https://recipes.tidymodels.org/reference/step_impute_bag.html
step_impute_knn
recipes::step_impute_knn()
https://recipes.tidymodels.org/reference/step_impute_knn.html
step_impute_linear
recipes::step_impute_linear()
https://recipes.tidymodels.org/reference/step_impute_linear.html
step_impute_lower
recipes::step_impute_lower()
https://recipes.tidymodels.org/reference/step_impute_lower.html
step_impute_mean
recipes::step_impute_mean()
https://recipes.tidymodels.org/reference/step_impute_mean.html
step_impute_median
recipes::step_impute_median()
https://recipes.tidymodels.org/reference/step_impute_median.html

step_impute_mode
recipes::step_impute_mode()
https://recipes.tidymodels.org/reference/step_impute_mode.html

step_impute_roll
recipes::step_impute_roll()
https://recipes.tidymodels.org/reference/step_impute_roll.html

Other Data Recipes: hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig(), pca_your_recipe()

Other Preprocessor: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_poly(),
hai_data_scale(), hai_data_transform(), hai_data_trig(), hai_earth_data_prepper(),
hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_poly_data_prepper(),
hai_svm_rbf_data_prepper(), hai_xgboost_data_prepper()

Examples

```r
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

date_seq <- seq.Date(from = as.Date("2013-01-01"), length.out = 100, by = "month")
val_seq <- rep(c(rnorm(9), NA), times = 10)
df_tbl <- tibble(
  date_col = date_seq,
  value = val_seq
)

rec_obj <- recipe(value ~ ., df_tbl)

hai_data_impute(
  .recipe_object = rec_obj,
  value,
  .type_of_imputation = "roll",
  .roll_statistic = median
)$impute_rec_obj %>%
get_juiced_data()
```

### Description

Takes in a recipe and will scale values using a selected recipe.
Usage

\texttt{hai\_data\_poly(.recipe\_object = NULL, ..., .p\_degree = 2)}

Arguments

\begin{itemize}
  \item \texttt{.recipe\_object}  The data that you want to process
  \item \texttt{...}  One or more selector functions to choose variables to be imputed. When used with \texttt{imp\_vars}, these dots indicate which variables are used to predict the missing data in each variable. See \texttt{selections()} for more details
  \item \texttt{.p\_degree}  The polynomial degree, an integer.
\end{itemize}

Details

This function will get your data ready for processing with many types of ml/ai models. This is intended to be used inside of the data processor and therefore is an internal function. This documentation exists to explain the process and help the user understand the parameters that can be set in the pre-processor function.

\texttt{recipes::step\_poly()}

Value

A list object

Author(s)

Steven P. Sanderson II, MPH

See Also

https://recipes.tidymodels.org/reference/step_poly.html

Other Data Recipes: \texttt{hai\_data\_impute()}, \texttt{hai\_data\_scale()}, \texttt{hai\_data\_transform()}, \texttt{hai\_data\_trig()}

Other Preprocessor: \texttt{hai\_c50\_data\_prepper()}, \texttt{hai\_cubist\_data\_prepper()}, \texttt{hai\_data\_impute()}

Examples

\begin{verbatim}
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

date_seq <- seq.Date(from = as.Date("2013-01-01"), length.out = 100, by = "month")
val_seq <- rep(rnorm(10, mean = 6, sd = 2), times = 10)
df_tbl <- tibble(
  date_col = date_seq,
  value = val_seq)
\end{verbatim}
rec_obj <- recipe(value ~ ., df_tbl)

hai_data_poly(
  .recipe_object = rec_obj,
  value
)$scale_rec_obj %>%
  get_juiced_data()

---

**Data Preprocessor - Scale/Normalize**

**Description**

Takes in a recipe and will scale values using a selected recipe. To call the recipe use a quoted argument like "scale" or "normalize".

**Usage**

```r
hai_data_scale(
  .recipe_object = NULL,
  ..., 
  .type_of_scale = "center",
  .range_min = 0,
  .range_max = 1,
  .scale_factor = 1
)
```

**Arguments**

- `.recipe_object` The data that you want to process
- `...` One or more selector functions to choose variables to be imputed. When used with imp_vars, these dots indicate which variables are used to predict the missing data in each variable. See selections() for more details.
- `.type_of_scale` This is a quoted argument and can be one of the following:
  - "center"
  - "normalize"
  - "range"
  - "scale"
- `.range_min` A single numeric value for the smallest value in the range. This defaults to 0.
- `.range_max` A single numeric value for the largest value in the range. This defaults to 1.
- `.scale_factor` A numeric value of either 1 or 2 that scales the numeric inputs by one or two standard deviations. By dividing by two standard deviations, the coefficients attached to continuous predictors can be interpreted the same way as with binary inputs. Defaults to 1. More in reference below.
Details

This function will get your data ready for processing with many types of ml/ai models.
This is intended to be used inside of the data processor and therefore is an internal function. This documentation exists to explain the process and help the user understand the parameters that can be set in the pre-processor function.

Value

A list object

Author(s)

Steven P. Sanderson II, MPH

References


See Also

step_center
recipes::step_center()
https://recipes.tidymodels.org/reference/step_center.html
step_normalize
recipes::step_normalize()
https://recipes.tidymodels.org/reference/step_normalize.html
step_range
recipes::step_range()
https://recipes.tidymodels.org/reference/step_range.html
step_scale
recipes::step_scale()
https://recipes.tidymodels.org/reference/step_scale.html

Other Data Recipes: hai_data_impute(), hai_data_poly(), hai_data_transform(), hai_data_trig(), pca_your_recipe()

Other Preprocessor: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_impute(), hai_data_poly(), hai_data_transform(), hai_data_trig(), hai_earth_data_prepper(), hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_poly_data_prepper(), hai_svm_rbf_data_prepper(), hai_xgboost_data_prepper()
Examples

suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

date_seq <- seq.Date(from = as.Date("2013-01-01"), length.out = 100, by = "month")
val_seq <- rep(rnorm(10, mean = 6, sd = 2), times = 10)
df_tbl <- tibble(
  date_col = date_seq,
  value = val_seq
)

rec_obj <- recipe(value ~ ., df_tbl)

hai_data_scale(
  .recipe_object = rec_obj,
  value,
  .type_of_scale = "center"
)$scale_rec_obj %>%
  get_juiced_data()

hai_data_transform  Data Preprocessor - Transformation Functions

Description

Takes in a recipe and will perform the desired transformation on the selected variable(s) using a selected recipe. To call the desired transformation recipe use a quoted argument like "boxcos", "bs" etc.

Usage

hai_data_transform(
  .recipe_object = NULL,
  ..., .type_of_scale = "log",
  .bc_limits = c(-5, 5),
  .bc_num_unique = 5,
  .bs_deg_free = NULL,
  .bs_degree = 3,
  .log_base = exp(1),
  .log_offset = 0,
  .logit_offset = 0,
  .ns_deg_free = 2,
  .rel_shift = 0,
  .rel_reverse = FALSE,
  .rel_smooth = FALSE,
  .yj_limits = c(-5, 5),
)
\[ y_j \text{num}_\text{unique} = 5 \]

## Arguments

- **recipe_object**: The data that you want to process
- **...**: One or more selector functions to choose variables to be imputed. When used with imp_vars, these dots indicate which variables are used to predict the missing data in each variable. See selections() for more details
- **type_of_scale**: This is a quoted argument and can be one of the following:
  - "boxcox"
  - "bs"
  - "log"
  - "logit"
  - "ns"
  - "relu"
  - "sqrt"
  - "yoejohnson"
- **bc_limits**: A length 2 numeric vector defining the range to compute the transformation parameter lambda.
- **bc_num_unique**: An integer to specify minimum required unique values to evaluate for a transformation
- **bs_deg_free**: The degrees of freedom for the spline. As the degrees of freedom for a spline increase, more flexible and complex curves can be generated. When a single degree of freedom is used, the result is a rescaled version of the original data.
- **bs_degree**: Degree of polynomial spline (integer).
- **log_base**: A numeric value for the base.
- **log_offset**: An optional value to add to the data prior to logging (to avoid log(0))
- **logit_offset**: A numeric value to modify values of the columns that are either one or zero. They are modified to be \( x - \text{offset} \) or \( \text{offset} \) respectively.
- **ns_deg_free**: The degrees of freedom for the natural spline. As the degrees of freedom for a natural spline increase, more flexible and complex curves can be generated. When a single degree of freedom is used, the result is a rescaled version of the original data.
- **rel_shift**: A numeric value dictating a translation to apply to the data.
- **rel_reverse**: A logical to indicate if the left hinge should be used as opposed to the right hinge.
- **rel_smooth**: A logical indicating if the softplus function, a smooth approximation to the rectified linear transformation, should be used.
- **yj_limits**: A length 2 numeric vector defining the range to compute the transformation parameter lambda.
- **yj_num_unique**: An integer where data that have less possible values will not be evaluated for a transformation.
Details

This function will get your data ready for processing with many types of ml/ai models.

This is intended to be used inside of the data processor and therefore is an internal function. This documentation exists to explain the process and help the user understand the parameters that can be set in the pre-processor function.

recipes::step_BoxCox()

Value

A list object

Author(s)

Steven P. Sanderson II, MPH

See Also

https://recipes.tidymodels.org/reference/step_BoxCox.html
recipes::step_bs()
https://recipes.tidymodels.org/reference/step_bs.html
recipes::step_log()
https://recipes.tidymodels.org/reference/step_log.html
recipes::step_logit()
https://recipes.tidymodels.org/reference/step_logit.html
recipes::step_ns()
https://recipes.tidymodels.org/reference/step_ns.html
recipes::step_relu()
https://recipes.tidymodels.org/reference/step_relu.html
recipes::step_sqrt()
https://recipes.tidymodels.org/reference/step_sqrt.html
recipes::step_YeoJohnson()
https://recipes.tidymodels.org/reference/step_YeoJohnson.html

Other Data Recipes: hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_trig(), pca_your_recipe()

Other Preprocessor: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_trig(), hai_earth_data_prepper(), hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_poly_data_prepper(), hai_svm_rbf_data_prepper(), hai_xgboost_data_prepper()
Examples

```r
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

date_seq <- seq.Date(from = as.Date("2013-01-01"), length.out = 100, by = "month")
val_seq <- rep(rnorm(10, mean = 6, sd = 2), times = 10)
df_tbl <- tibble(
  date_col = date_seq,
  value = val_seq
)

rec_obj <- recipe(value ~ ., df_tbl)

hai_data_transform(
  .recipe_object = rec_obj,
  value,
  .type_of_scale = "log"
)$scale_rec_obj %>%
get_juiced_data()
```

---

**hai_data_trig**  
*Data Preprocessor - Trigonometric Functions*

**Description**

Takes in a recipe and will scale values using a selected recipe. To call the recipe use a quoted argument like "sinh", "cosh" or "tanh".

**Usage**

```r
hai_data_trig(
  .recipe_object = NULL,
  ...,
  .type_of_scale = "sinh",
  .inverse = FALSE
)
```

**Arguments**

- `.recipe_object`  The data that you want to process
- `...`  One or more selector functions to choose variables to be imputed. When used with `imp_vars`, these dots indicate which variables are used to predict the missing data in each variable. See `selections()` for more details
- `.type_of_scale`  This is a quoted argument and can be one of the following:
  - "sinh"
  - "cosh"
  - "tanh"
- `.inverse`  A logical: should the inverse function be used? Default is FALSE
Details

This function will get your data ready for processing with many types of ml/ai models.

This is intended to be used inside of the data processor and therefore is an internal function. This documentation exists to explain the process and help the user understand the parameters that can be set in the pre-processor function.

\texttt{recipes::step_hyperbolic()}

Value

A list object

Author(s)

Steven P. Sanderson II, MPH

See Also

https://recipes.tidymodels.org/reference/step_hyperbolic.html

Other Data Recipes: \texttt{hai_data_impute()}, \texttt{hai_data_poly()}, \texttt{hai_data_scale()}, \texttt{hai_data_transform()}, \texttt{pca_your_recipe()}

Other Preprocessor: \texttt{hai_c50_data_prepper()}, \texttt{hai_cubist_data_prepper()}, \texttt{hai_data_impute()}, \texttt{hai_data_poly()}, \texttt{hai_data_scale()}, \texttt{hai_data_transform()}, \texttt{hai_earth_data_prepper()}, \texttt{hai_glmnet_data_prepper()}, \texttt{hai_knn_data_prepper()}, \texttt{hai_ranger_data_prepper()}, \texttt{hai_svm_poly_data_prepper()}, \texttt{hai_svm_rbf_data_prepper()}, \texttt{hai_xgboost_data_prepper()}

Examples

```r
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

date_seq <- seq.Date(from = as.Date("2013-01-01"), length.out = 100, by = "month")
val_seq <- rep(rnorm(10, mean = 6, sd = 2), times = 10)
df_tbl <- tibble(
  date_col = date_seq,
  value = val_seq
)

c <- recipe(value ~ ., df_tbl)
hai_data_trig(
  .recipe_object = c,
  value,
  .type_of_scale = "sinh"
)$scale_rec_obj %>%
get_juiced_data()
```
Description
Default classification metric sets from yardstick

Usage
hai_default_classification_metric_set()

Details
Default classification metric sets from yardstick

Value
A yardstick metric set tibble

Author(s)
Steven P. Sanderson II, MPH

See Also
Other Default Metric Sets: hai_default_regression_metric_set()

Examples
hai_default_classification_metric_set()

Description
Default regression metric sets from yardstick

Usage
hai_default_regression_metric_set()
Details

Default regression metric sets from yardstick

Value

A yardstick metric set tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Default Metric Sets: `hai_default_classification_metric_set()`

Examples

`hai_default_regression_metric_set()`

Description

this will produce a ggplot2 or plotly histogram plot of the density information provided from the `hai_get_density_data_tbl` function.

Usage

```r
hai_density_hist_plot(
  .data, .dist_name_col = distribution, .value_col = dist_data,
  .alpha = 0.382, .interactive = FALSE
)
```

Arguments

- `.data`: The data that is produced from using `hai_get_density_data_tbl`
- `.dist_name_col`: The column that has the distribution name, should be `distribution` and that is set as the default.
- `.value_col`: The column that contains the x values that comes from the `hai_get_density_data_tbl` function.
- `.alpha`: The alpha parameter for ggplot
- `.interactive`: This is a Boolean for TRUE/FALSE and is defaulted to FALSE. TRUE will produce a plotly plot.
Details
This will produce a histogram of the density information that is produced from the function `hai_get_density_data_tbl`. It will look for an attribute from the `.data` param to ensure the function was used.

Value
A plot, either ggplot2 or plotly

Author(s)
Steven P. Sanderson II, MPH

See Also
Other Distribution Plots: `hai_density_plot()`, `hai_density_qq_plot()`

Examples
```r
library(dplyr)

df <- hai_scale_zero_one_vec(.x = mtcars$mpg) %>%
  hai_distribution_comparison_tbl()

dist_data_tbl <- hai_get_dist_data_tbl(df)

hai_density_hist_plot(
  .data = dist_data_tbl,
  .dist_name_col = distribution,
  .value_col = dist_data,
  .alpha = 0.5,
  .interactive = FALSE
)
```

Description
This will produce a ggplot2 or plotly histogram plot of the density information provided from the `hai_get_density_data_tbl` function.

Usage
```r
hai_density_plot(
  .data,
  .dist_name_col,
  .x_col,
)```
Arguments

.data The data that is produced from using `hai_get_density_data_tbl`
.dist_name_col The column that has the distribution name, should be distribution and that is set as the default.
.x_col The x value from the tidied density object.
.y_col The y value from the tidied density object.
.size The size parameter for ggplot.
.alpha The alpha parameter for ggplot.
.interactive This is a Boolean for TRUE/FALSE and is defaulted to FALSE. TRUE will produce a plotly plot.

Details

This will produce a density plot of the density information that is produced from the function `hai_get_density_data_tbl`. It will look for an attribute from the .data param to ensure the function was used.

Value

A plot, either ggplot2 or plotly

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Distribution Plots: `hai_density_hist_plot()`, `hai_density_qq_plot()`

Examples

```r
library(dplyr)

df <- hai_scale_zero_one_vec(.x = mtcars$mpg) %>%
  hai_distribution_comparison_tbl()

tidy_density_tbl <- hai_get_density_data_tbl(df)

hai_density_plot(
  .data = tidy_density_tbl,
  .dist_name_col = distribution,
  .x_col = x,
  .y_col, .size = 1, .alpha = 0.382, .interactive = FALSE)
```
hai_density_qq_plot

```r
.y_col = y,
.alpha = 0.5,
.interactive = FALSE
)
```

---

**Description**

This will produce a ggplot or plotly histogram plot of the density information provided from the `hai_get_density_data_tbl` function.

**Usage**

```r
hai_density_qq_plot(
  .data,
  .dist_name_col = distribution,
  .x_col = x,
  .y_col = y,
  .size = 1,
  .alpha = 0.382,
  .interactive = FALSE
)
```

**Arguments**

- `.data` The data that is produced from using `hai_get_density_data_tbl`
- `.dist_name_col` The column that has the distribution name, should be distribution and that is set as the default.
- `.x_col` The column that contains the x values that comes from the `hai_get_density_data_tbl` function.
- `.y_col` The column that contains the y values that comes from the `hai_get_density_data_tbl` function.
- `.size` The size parameter for ggplot
- `.alpha` The alpha parameter for ggplot
- `.interactive` This is a Boolean fo TRUE/FALSE and is defaulted to FALSE. TRUE will produce a plotly plot.

**Details**

This will produce a qq plot of the density information that is produced from the function `hai_get_density_data_tbl`. It will look for an attribute from the `.data` param to ensure the function was used.
Value

A plot, either ggplot2 or plotly

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Distribution Plots: \texttt{hai\_density\_hist\_plot()}, \texttt{hai\_density\_plot()}

Examples

library(dplyr)

df <- hai\_scale\_zero\_one\_vec(.x = mtcars\$mpg) %>%
    hai\_distribution\_comparison\_tbl()

tidy\_density\_tbl <- hai\_get\_density\_data\_tbl(df)

hai\_density\_qq\_plot(
    .data = tidy\_density\_tbl,
    .dist\_name\_col = distribution,
    .x\_col = x,
    .y\_col = y,
    .size = 1,
    .alpha = 0.5,
    .interactive = FALSE
)

\begin{verbatim}
# hai\_distribution\_comparison\_tbl

\textbf{Compare Data Against Distributions}
\end{verbatim}

Description

This function will attempt to get some key information on the data you pass to it. It will also automatically normalize the data from 0 to 1. This will not change the distribution just its scale in order to make sure that many different types of distributions can be fit to the data, which should help identify what the distribution of the passed data could be.

The resulting output has attributes added to it that get used in other functions that are meant to compliment each other.

This function will automatically pass the \texttt{x} parameter to \texttt{hai\_skewness\_vec()} and \texttt{hai\_kurtosis\_vec()} in order to help create the random data from the distributions.

The distributions that can be chosen from are:

\begin{verbatim}
Distribution R stats::dist
\end{verbatim}
hai_distribution_comparison_tbl

<table>
<thead>
<tr>
<th>Normal</th>
<th>Rnorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>Runif</td>
</tr>
<tr>
<td>Exponential</td>
<td>Rexp</td>
</tr>
<tr>
<td>Logistic</td>
<td>Rlogis</td>
</tr>
<tr>
<td>Beta</td>
<td>Rbeta</td>
</tr>
<tr>
<td>Lognormal</td>
<td>Rlnorm</td>
</tr>
<tr>
<td>Gamma</td>
<td>Rgamma</td>
</tr>
<tr>
<td>Weibull</td>
<td>Weibull</td>
</tr>
<tr>
<td>Chi-square</td>
<td>Rchisq</td>
</tr>
<tr>
<td>Cauchy</td>
<td>Rcauchy</td>
</tr>
<tr>
<td>Hypergeometric</td>
<td>Rhyper</td>
</tr>
<tr>
<td>F</td>
<td>Rf</td>
</tr>
<tr>
<td>Poisson</td>
<td>Rpois</td>
</tr>
</tbody>
</table>

**Usage**

```r
hai_distribution_comparison_tbl(
  .x,
  .distributions = c("gamma", "beta"),
  .normalize = TRUE
)
```

**Arguments**

- `.x` The numeric vector to analyze.
- `.distributions` A character vector of distributions to check. For example, c("gamma", "beta")
- `.normalize` A boolean value of TRUE/FALSE, the default is TRUE. This will normalize the data using the `hai_scale_zero_one_vec` function.

**Details**

Get information on the empirical distribution of your data along with generated densities of other distributions. This information is in the resulting tibble that is generated. Three columns will generate, Distribution, from the param .distributions, dist_data which is a list vector of density values passed to the underlying stats r distribution function, and density_data, which is the dist_data column passed to list(stats::density(unlist(dist_data)))

This has the effect of giving you the desired vector that can be used in resultant plots (dist_data) or you can interact with the density object itself.

If the skewness of the distribution is negative, then for the gamma and beta distributions the skew is set equal to the kurtosis and the kurtosis is set equal to sqrt((skew)^2)

**Value**

A tibble.

**Author(s)**

Steven P. Sanderson II, MPH
hai_earth_data_prepper

Prep Data for Earth - Recipe

Description
Automatically prep a data.frame/tibble for use in the Earth algorithm.

Usage
hai_earth_data_prepper(.data, .recipe_formula)

Arguments
.data The data that you are passing to the function. Can be any type of data that is accepted by the data parameter of the recipes::recip() function.
.recipe_formula The formula that is going to be passed. For example if you are using the diamonds data then the formula would most likely be something like price ~ .

Details
This function will automatically prep your data.frame/tibble for use in the Earth algorithm. The Earth algorithm is for classification and regression.
This function will output a recipe specification.

Value
A recipe object

Author(s)
Steven P. Sanderson II, MPH
See Also

http://uc-r.github.io/mars

Other Preprocessor: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig(), hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_poly_data_prepper(), hai_svm_rbf_data_prepper(), hai_xgboost_data_prepper()

Other Earth: hai_auto_earth()

Examples

library(ggplot2)

# Regression
hai_earth_data_prepper(.data = diamonds, .recipe_formula = price ~ .)
reg_obj <- hai_earth_data_prepper(diamonds, price ~ .)
get_juiced_data(reg_obj)

# Classification
hai_earth_data_prepper(Titanic, Survived ~ .)
cla_obj <- hai_earth_data_prepper(Titanic, Survived ~ .)
get_juiced_data(cla_obj)

hai_fourier_augment Augment Function Fourier

Description

Takes a numeric vector(s) or date and will return a tibble of one of the following:

- "sin"
- "cos"
- "sincos"
- c("sin","cos","sincos")

Usage

hai_fourier_augment(
  .data,
  .value,
  .period,
  .order,
  .names = "auto",
  .scale_type = c("sin", "cos", "sincos")
)
Arguments

.data The data being passed that will be augmented by the function.
.value This is passed `rlang::enquo()` to capture the vectors you want to augment.
.period The number of observations that complete a cycle
.order The fourier term order
.names The default is "auto"
.scale_type A character of one of the following: "sin","cos", or sincos" All can be passed by setting the param equal to `c("sin","cos","sincos")`

Details

Takes a numeric vector or date and will return a vector of one of the following:

- "sin"
- "cos"
- "sincos"
- `c("sin","cos","sincos")`

This function is intended to be used on its own in order to add columns to a tibble.

Value

A augmented tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function:

- `hai_fourier_discrete_augment()`, `hai_hyperbolic_augment()`, `hai_polynomial_augment()`,
- `hai_scale_zero_one_augment()`, `hai_scale_zscore_augment()`, `hai Winsorized move augment()`,
- `hai Winsorized truncate augment()`

Examples

suppressPackageStartupMessages(library(dplyr))

len_out <- 10
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)
hai_fourier_discrete_augment

Augment Function Fourier Discrete

Description
Takes a numeric vector(s) or date and will return a tibble of one of the following:

- "sin"
- "cos"
- "sincos"
- c("sin","cos","sincos") When either of these values falls below zero, then zero else one

Usage

```r
hai_fourier_discrete_augment(
  .data,
  .value,
  .period,
  .order,
  .names = "auto",
  .scale_type = c("sin","cos","sincos")
)
```

Arguments

- `.data` The data being passed that will be augmented by the function.
- `.value` This is passed `rlang::enquo()` to capture the vectors you want to augment.
- `.period` The number of observations that complete a cycle
- `.order` The fourier term order
- `.names` The default is "auto"
- `.scale_type` A character of one of the following: "sin","cos", or sincos" All can be passed by setting the param equal to c("sin","cos","sincos")

Details
Takes a numeric vector or a date and will return a vector of one of the following:

- "sin"
- "cos"
- "sincos"
- c("sin","cos","sincos")

This function is intended to be used on its own in order to add columns to a tibble.
Value

A augmented tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function: `hai_fourier_augment()`, `hai_hyperbolic_augment()`, `hai_polynomial_augment()`, `hai_scale_zero_one_augment()`, `hai_scale_zscore_augment()`, `haiWinsorized_move_augment()`, `haiWinsorized_truncate_augment()`

Examples

```r
suppressPackageStartupMessages(library(dplyr))

len_out <- 24
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

hai_fourier_discrete_augment(data_tbl, b, .period = 2 * 12, .order = 1, .scale_type = "sin")
hai_fourier_discrete_augment(data_tbl, b, .period = 2 * 12, .order = 1, .scale_type = "cos")
```

**Description**

Takes a numeric vector or date and will return a vector of one of the following:

- "sin"
- "cos"
- "sincos" This will do value = sin(x) * cos(x) When either of these values falls below zero, then zero else one
hai_fourier_discrete_vec

Usage

hai_fourier_discrete_vec(
  .x,
  .period,
  .order,
  .scale_type = c("sin", "cos", "sincos")
)

Arguments

.x A numeric vector
.period The number of observations that complete a cycle
.order The fourier term order
.scale_type A character of one of the following: "sin","cos","sincos"

Details

Takes a numeric vector or date and will return a vector of one of the following:

- "sin"
- "cos"
- "sincos"

The internal calculation is straightforward:

- \( \sin = \sin(2 \pi h x) \), where \( h = \frac{.order}{.period} \)
- \( \cos = \cos(2 \pi h x) \), where \( h = \frac{.order}{.period} \)
- \( \text{sincos} = \sin(2 \pi h x) \times \cos(2 \pi h x) \) where \( h = \frac{.order}{.period} \)

This function can be used on its own. It is also the basis for the function hai_fourier_discrete_augment().

Value

A numeric vector of 1's and 0's

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Vector Function: hai_fourier_vec(), hai_hyperbolic_vec(), hai_kurtosis_vec(), hai_scale_zero_one_vec(), hai_scale_zscore_vec(), hai_skewness_vec(), haiWinsorized_move_vec(), haiWinsorized_truncate_vec()
Examples

```r
suppressPackageStartupMessages(library(dplyr))

len_out <- 24
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

vec_1 <- hai_fourier_discrete_vec(data_tbl$a, .period = 12, .order = 1, .scale_type = "sin")
vec_2 <- hai_fourier_discrete_vec(data_tbl$a, .period = 12, .order = 1, .scale_type = "cos")
vec_3 <- hai_fourier_discrete_vec(data_tbl$a, .period = 12, .order = 1, .scale_type = "sincos")

plot(data_tbl$b)
lines(vec_1, col = "blue")
lines(vec_2, col = "red")
lines(vec_3, col = "green")
```

**hai_fourier_vec**

*Vector Function Fourier*

**Description**

Takes a numeric vector and will return a vector of one of the following:

- "sin"
- "cos"
- "sincos" This will do value = sin(x) * cos(x)

**Usage**

```r
 hai_fourier_vec(.x, .period, .order, .scale_type = c("sin", "cos", "sincos"))
```

**Arguments**

- `.x` A numeric vector
- `.period` The number of observations that complete a cycle
- `.order` The fourier term order
- `.scale_type` A character of one of the following: "sin","cos","sincos"
Details

Takes a numeric vector and will return a vector of one of the following:
- "sin"
- "cos"
- "sincos"

The internal calculation is straightforward:
- $\sin = \sin(2 \pi h x)$, where $h = .order/.period$
- $\cos = \cos(2 \pi h x)$, where $h = .order/.period$
- $\text{sincos} = \sin(2 \pi h x) \times \cos(2 \pi h x)$ where $h = .order/.period$

This function can be used on its own. It is also the basis for the function `hai_fourier_augment()`.

Value

A numeric vector

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Vector Function: `hai_fourier_discrete_vec()`, `hai_hyperbolic_vec()`, `hai_kurtosis_vec()`, `hai_scale_zero_one_vec()`, `hai_scale_zscore_vec()`, `hai_skewness_vec()`, `hai_winsorized_move_vec()`, `hai_winsorized_truncate_vec()`

Examples

```r
suppressPackageStartupMessages(library(dplyr))

len_out <- 25
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

vec_1 <- hai_fourier_vec(data_tbl$b, .period = 12, .order = 1, .scale_type = "sin")
vec_2 <- hai_fourier_vec(data_tbl$b, .period = 12, .order = 1, .scale_type = "cos")
vec_3 <- hai_fourier_vec(data_tbl$date_col, .period = 12, .order = 1, .scale_type = "sincos")

plot(data_tbl$b)
lines(vec_1, col = "blue")
lines(vec_2, col = "red")
lines(vec_3, col = "green")
```
hai_get_density_data_tbl

*Get Density Data Helper*

**Description**

This function will return a tibble that can either be nested/unnested, and grouped or un-grouped. The `.data` argument must be the output of the `hai_distribution_comparison_tbl()` function.

**Usage**

```r
hai_get_density_data_tbl(.data, .unnest = TRUE, .group_data = TRUE)
```

**Arguments**

- `.data` The data from the `hai_distribution_comparison_tbl()` function as this function looks for an attribute of `hai_dist_compare_tbl`
- `.unnest` Should the resulting tibble be un-nested, a Boolean value TRUE/FALSE. The default is TRUE
- `.group_data` Should the resulting tibble be grouped, a Boolean value TRUE/FALSE. The default is FALSE

**Details**

This function expects to take the output of the `hai_distribution_comparison_tbl()` function. It returns a tibble of the tidy density data.

**Value**

A tibble.

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Distribution Functions: `hai_distribution_comparison_tbl()`, `hai_get_dist_data_tbl()`

**Examples**

```r
library(dplyr)

df <- hai_scale_zero_one_vec(.x = mtcars$mpg) %>%
  hai_distribution_comparison_tbl()
hai_get_density_data_tbl(df)
```
**hai_get_dist_data_tbl**  
*Get Distribution Data Helper*

**Description**

This function will return a tibble that can either be nested/unnested, and grouped or ungrouped. The .data argument must be the output of the hai_distribution_comparison_tbl() function.

**Usage**

```r
hai_get_dist_data_tbl(.data, .unnest = TRUE, .group_data = FALSE)
```

**Arguments**

- `.data` The data from the hai_distribution_comparison_tbl() function as this function looks for a class of 'hai_dist_data'
- `.unnest` Should the resulting tibble be unnested, a boolean value TRUE/FALSE. The default is TRUE
- `.group_data` Should the resulting tibble be grouped, a boolean value TRUE/FALSE. The default is FALSE

**Details**

This function expects to take the output of the hai_distribution_comparison_tbl() function. It returns a tibble of the distribution and the randomly generated data produced from the associated stats r function like rnorm.

**Value**

A tibble.

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Distribution Functions: hai_distribution_comparison_tbl(), hai_get_density_data_tbl()

**Examples**

```r
library(dplyr)

df <- hai_scale_zero_one_vec(.x = mtcars$mpg) %>%
    hai_distribution_comparison_tbl()
hai_get_dist_data_tbl(df)
```
Prep Data for glmnet - Recipe

Description

Automatically prep a data.frame/tibble for use in the glmnet algorithm.

Usage

hai_glmnet_data_prepper(.data, .recipe_formula)

Arguments

.data The data that you are passing to the function. Can be any type of data that is accepted by the data parameter of the recipes::recip() function.

.recipe_formula The formula that is going to be passed. For example if you are using the iris data then the formula would most likely be something like Species ~ .

Details

This function will automatically prep your data.frame/tibble for use in the glmnet algorithm. It expects data to be presented in a certain fashion.

This function will output a recipe specification.

Value

A recipe object

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Preprocessor: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig(), hai_earth_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_poly_data_prepper(), hai_svm_rbf_data_prepper(), hai_xgboost_data_prepper()

Other knn: hai_knn_data_prepper()
### hai_histogram_facet_plot

#### Description

This function expects a data.frame/tibble and will return a faceted histogram.

#### Usage

```r
hai_histogram_facet_plot(
  .data,
  .bins = 10,
  .scale_data = FALSE,
  .ncol = 5,
  .fct_reorder = FALSE,
  .fct_rev = FALSE,
  .fill = "steelblue",
  .color = "white",
  .scale = "free",
  .interactive = FALSE
)
```

#### Arguments

- `.data`  
  The data you want to pass to the function.
- `.bins`  
  The number of bins for the histograms.
- `.scale_data`  
  This is a boolean set to FALSE. TRUE will use `hai_scale_zero_one_vec()` to `[0, 1]` scale the data.
- `.ncol`  
  The number of columns for the facet_warp argument.
- `.fct_reorder`  
  Should the factor column be reordered? TRUE/FALSE, default of FALSE
- `.fct_rev`  
  Should the factor column be reversed? TRUE/FALSE, default of FALSE
- `.fill`  
  Default is steelblue
- `.color`  
  Default is 'white'
- `.scale`  
  Default is 'free'
- `.interactive`  
  Default is FALSE, TRUE will produce a plotly plot.
**Details**
Takes in a data.frame/tibble and returns a faceted histogram.

**Value**
A ggplot or plotly plot

**Author(s)**
Steven P. Sanderson II, MPH

**Examples**

```r
hai_histogram_facet_plot(.data = iris)
hai_histogram_facet_plot(.data = iris, .scale_data = TRUE)
```

---

**hai_hyperbolic_augment**

*Augment Function Hyperbolic*

**Description**
Takes a numeric vector(s) or date and will return a tibble of one of the following:

- "sin"
- "cos"
- "tan"
- "sincos"
- c("sin", "cos", "tan", "sincos")

**Usage**

```r
hai_hyperbolic_augment(
  .data,  
  .value, 
  .names = "auto", 
  .scale_type = c("sin", "cos", "tan", "sincos")
)
```

**Arguments**

- `.data` The data being passed that will be augmented by the function.
- `.value` This is passed `rlang::enquo()` to capture the vectors you want to augment.
- `.names` The default is "auto"
- `.scale_type` A character of one of the following: "sin", "cos", "tan", "sincos" All can be passed by setting the param equal to c("sin", "cos", "tan", "sincos")
Details

Takes a numeric vector or date and will return a vector of one of the following:

- "sin"
- "cos"
- "tan"
- "sincos"
- c("sin","cos","tan", "sincos")

This function is intended to be used on its own in order to add columns to a tibble.

Value

A augmented tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function: hai_fourier_augment(), hai_fourier_discrete_augment(), hai_polynomial_augment(), hai_scale_zero_one_augment(), hai_scale_zscore_augment(), haiWinsorized_move_augment(), haiWinsorized_truncate_augment()

Examples

suppressPackageStartupMessages(library(dplyr))

len_out <- 10
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

hai_hyperbolic_augment(data_tbl, b, .scale_type = "sin")
hai_hyperbolic_augment(data_tbl, b, .scale_type = "tan")
### hai_hyperbolic_vec

**Vector Function Hyperbolic**

**Description**

Takes a numeric vector and will return a vector of one of the following:

- "sin"
- "cos"
- "tan"
- "sincos" This will do value = sin(x) * cos(x)

**Usage**

```r
hai_hyperbolic_vec(.x, .scale_type = c("sin", "cos", "tan", "sincos"))
```

**Arguments**

- `.x` A numeric vector
- `.scale_type` A character of one of the following: "sin", "cos", "tan", "sincos"

**Details**

Takes a numeric vector and will return a vector of one of the following:

- "sin"
- "cos"
- "tan"
- "sincos"

This function can be used on its own. It is also the basis for the function `hai_hyperbolic_augment()`.

**Value**

A numeric vector

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Vector Function: `hai_fourier_discrete_vec()`, `hai_fourier_vec()`, `hai_kurtosis_vec()`, `hai_scale_zero_one_vec()`, `hai_scale_zscore_vec()`, `hai_skewness_vec()`, `hai_winsorized_move_vec()`, `hai_winsorized_truncate_vec()`
Examples

suppressPackageStartupMessages(library(dplyr))

len_out <- 25
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

vec_1 <- hai_hyperbolic_vec(data_tbl$b, .scale_type = "sin")
vec_2 <- hai_hyperbolic_vec(data_tbl$b, .scale_type = "cos")
vec_3 <- hai_hyperbolic_vec(data_tbl$b, .scale_type = "sincos")

plot(data_tbl$b)
lines(vec_1, col = "blue")
lines(vec_2, col = "red")
lines(vec_3, col = "green")

hai_kmeans_automl

Automatic K-Means H2O

Description

This is a wrapper around the h2o::h2o.kmeans() function that will return a list object with a lot of useful and easy to use tidy style information.

Usage

hai_kmeans_automl(
  .data,
  .split_ratio = 0.8,
  .seed = 1234,
  .centers = 10,
  .standardize = TRUE,
  .print_model_summary = TRUE,
  .predictors,
  .categorical_encoding = "auto",
  .initialization_mode = "Furthest",
  .max_iterations = 100
)
Arguments

.data The data that is to be passed for clustering.
.split_ratio The ratio for training and testing splits.
.seed The default is 1234, but can be set to any integer.
.centers The default is 1. Specify the number of clusters (groups of data) in a data set.
.standardize The default is set to TRUE. When TRUE all numeric columns will be set to zero mean and unit variance.
.print_model_summary This is a boolean and controls if the model summary is printed to the console. The default is TRUE.
.predictors This must be in the form of c("column_1", "column_2", ..., "column_n")
.categorical_encoding Can be one of the following:
  • "auto"
  • "enum"
  • "one_hot_explicit"
  • "binary"
  • "eigen"
  • "label_encoder"
  • "sort_by_response"
  • "enum_limited"
.initialization_mode This can be one of the following:
  • "Random"
  • "Furthest (default)"
  • "PlusPlus"
.max_iterations The default is 100. This specifies the number of training iterations

Value

A list object

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Kmeans: hai_kmeans_automl_predict(), hai_kmeans_mapped_tbl(), hai_kmeans_obj(), hai_kmeans_scree_data_tbl(), hai_kmeans_scree_plt(), hai_kmeans_tidy_tbl(), hai_kmeans_user_item_tbl()
Examples

```r
## Not run:
h2o.init()
output <- hai_kmeans_automl(
  .data = iris,
  .standardize = FALSE
)
h2o.shutdown()

## End(Not run)
```

Description

This is a wrapper around the `h2o::h2o.predict()` function that will return a list object with a lot of useful and easy to use tidy style information.

Usage

```r
hai_kmeans_automl_predict(.input)
```

Arguments

- `.input` This is the output of the `hai_kmeans_automl()` function.

Details

This function will internally take in the output assigned from the `hai_kmeans_automl()` function only and return a list of useful information. The items that are returned are as follows:

1. `prediction` - The h2o dataframe of predictions
2. `prediction_tbl` - The h2o predictions in tibble format
3. `valid_tbl` - The validation data in tibble format
4. `pred_full_tbl` - The entire validation set with the predictions attached using `base::cbind()`. The predictions are in a column called `predicted_cluster` and are in the formate of a factor using `forcats::as_factor()`

Value

A list object
Author(s)

Steven P. Sanderson II, MPH

See Also

Other Kmeans: hai_kmeans_automl(), hai_kmeans_mapped_tbl(), hai_kmeans_obj(), hai_kmeans_scree_data_tbl(), hai_kmeans_scree_plt(), hai_kmeans_tidy_tbl(), hai_kmeans_user_item_tbl()

Examples

```r
## Not run:
h2o.init()

output <- hai_kmeans_automl(
  .data = iris,
  .standardize = FALSE
)

pred <- hai_kmeans_automl_predict(output)

h2o.shutdown()

## End(Not run)
```

hai_kmeans_mapped_tbl  K-Means Mapping Function

Description

Create a tibble that maps the hai_kmeans_obj() using purrr::map() to create a nested data.frame/tibble that holds n centers. This tibble will be used to help create a scree plot.

Usage

```r
hai_kmeans_mapped_tbl(.data, .centers = 15)
kmeans_mapped_tbl(.data, .centers = 15)
```

Arguments

- `.data`  You must have a tibble in the working environment from the hai_kmeans_user_item_tbl()
- `.centers`  How many different centers do you want to try

Details

Takes in a single parameter of .centers. This is used to create the tibble and map the hai_kmeans_obj() function down the list creating a nested tibble.
**Value**

A nested tibble

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

https://en.wikipedia.org/wiki/Scree_plot

Other Kmeans: `hai_kmeans_automl_predict()`, `hai_kmeans_automl()`, `hai_kmeans_obj()`, `hai_kmeans_scree_data_tbl()`, `hai_kmeans_scree_plt()`, `hai_kmeans_tidy_tbl()`,
`hai_kmeans_user_item_tbl()`

**Examples**

```r
library(healthyR.data)
library(dplyr)

data_tbl <- healthyR_data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?") %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()

ui_tbl <- hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
)

hai_kmeans_mapped_tbl(ui_tbl)
```

---

**Description**

Takes the output of the `hai_kmeans_user_item_tbl()` function and applies the k-means algorithm to it using `stats::kmeans()`

**Usage**

```r
hai_kmeans_obj(.data, .centers = 5)

kmeans_obj(.data, .centers = 5)
```
Arguments

.data  The data that gets passed from `hai_kmeans_user_item_tbl()`
.centers  How many initial centers to start with

Details

Uses the `stats::kmeans()` function and creates a wrapper around it.

Value

A stats k-means object

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Kmeans: `hai_kmeans_automl_predict()`, `hai_kmeans_automl()`, `hai_kmeans_mapped_tbl()`, `hai_kmeans_scree_data_tbl()`, `hai_kmeans_scree_plt()`, `hai_kmeans_tidy_tbl()`, `hai_kmeans_user_item_tbl()`

Examples

```r
library(healthyR.data)
library(dplyr)

data_tbl <- healthyR_data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?") %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()

hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
) %>%
  hai_kmeans_obj()
```
hai_kmeans_scree_data_tbl

K-Means Scree Plot Data Table

Description
Take data from the `hai_kmeans_mapped_tbl()` and unnest it into a tibble for inspection and for use in the `hai_kmeans_scree_plt()` function.

Usage
```r
hai_kmeans_scree_data_tbl(.data)
kmeans_scree_data_tbl(.data)
```

Arguments
- `.data` You must have a tibble in the working environment from the `hai_kmeans_mapped_tbl()`

Details
Takes in a single parameter of `.data` from `hai_kmeans_mapped_tbl()` and transforms it into a tibble that is used for `hai_kmeans_scree_plt()`. It will show the values (tot.withinss) at each center.

Value
A nested tibble

Author(s)
Steven P. Sanderson II, MPH

See Also
Other Kmeans: `hai_kmeans_automl_predict()`, `hai_kmeans_automl()`, `hai_kmeans_mapped_tbl()`, `hai_kmeans_obj()`, `hai_kmeans_scree_plt()`, `hai_kmeans_tidy_tbl()`, `hai_kmeans_user_item_tbl()`

Examples
```r
library(healthyR.data)
library(dplyr)

data_tbl <- healthyR.data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?") %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()
```
ui_tbl <- hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
)

kmm_tbl <- hai_kmeans_mapped_tbl(ui_tbl)

hai_kmeans_scree_data_tbl(kmm_tbl)

---

**hai_kmeans_scree_plt**  
 *K-Means Scree Plot*

---

**Description**
Create a scree-plot from the `hai_kmeans_mapped_tbl()` function.

**Usage**

```r
hai_kmeans_scree_plt(.data)
kmeans_scree_plt(.data)
hai_kmeans_scree_plot(.data)
```

**Arguments**

- `.data`  
  The data from the `hai_kmeans_mapped_tbl()` function

**Details**
Outputs a scree-plot

**Value**
A ggplot2 plot

**Author(s)**
Steven P. Sanderson II, MPH

**See Also**

https://en.wikipedia.org/wiki/Scree_plot

Other Kmeans: `hai_kmeans_automl_predict()`, `hai_kmeans_automl()`, `hai_kmeans_mapped_tbl()`, `hai_kmeans_obj()`, `hai_kmeans_scree_data_tbl()`, `hai_kmeans_tidy_tbl()`, `hai_kmeans_user_item_tbl()`
Examples

```r
library(healthyR.data)
library(dplyr)

data_tbl <- healthyR_data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?" ) %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()

ui_tbl <- hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
)

kmm_tbl <- hai_kmeans_mapped_tbl(ui_tbl)

hai_kmeans_scree_plt(.data = kmm_tbl)
```

---

**K-Means Object Tidy Functions**

### Description

K-Means tidy functions

### Usage

```r
hai_kmeans_tidy_tbl(.kmeans_obj, .data, .tidy_type = "tidy")

kmeans_tidy_tbl(.kmeans_obj, .data, .tidy_type = "tidy")
```

### Arguments

- `.kmeans_obj` A `stats::kmeans()` object
- `.data` The user item tibble created from `hai_kmeans_user_item_tbl()`
- `.tidy_type` "tidy","glance", or "augment"

### Details

Takes in a k-means object and its associated user item tibble and then returns one of the items asked for. Either: `broom::tidy()`, `broom::glance()` or `broom::augment()`. The function defaults to `broom::tidy()`. 
**Value**

A tibble

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Kmeans: hai_kmeans_automl_predict(), hai_kmeans_automl(), hai_kmeans_mapped_tbl(), hai_kmeans_obj(), hai_kmeans_scree_data_tbl(), hai_kmeans_scree_plt(), hai_kmeans_user_item_tbl()

**Examples**

```r
data_tbl <- healthyR_data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?") %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()

uit_tbl <- hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
)

km_obj <- hai_kmeans_obj(uit_tbl)

hai_kmeans_tidy_tbl(
  .kmeans_obj = km_obj,
  .data = uit_tbl,
  .tidy_type = "augment"
)

hai_kmeans_tidy_tbl(
  .kmeans_obj = km_obj,
  .data = uit_tbl,
  .tidy_type = "glance"
)

hai_kmeans_tidy_tbl(
  .kmeans_obj = km_obj,
  .data = uit_tbl,
  .tidy_type = "tidy"
)
```


Description

Takes in a data.frame/tibble and transforms it into an aggregated/normalized user-item tibble of proportions. The user will need to input the parameters for the rows/user and the columns/items.

Usage

```r
hai_kmeans_user_item_tbl(.data, .row_input, .col_input, .record_input)
```

```r
kmeans_user_item_tbl(.data, .row_input, .col_input, .record_input)
```

Arguments

- `.data` The data that you want to transform
- `.row_input` The column that is going to be the row (user)
- `.col_input` The column that is going to be the column (item)
- `.record_input` The column that is going to be summed up for the aggregation and normalization process.

Details

This function should be used before using a k-mean model. This is commonly referred to as a user-item matrix because "users" tend to be on the rows and "items" (e.g. orders) on the columns. You must supply a column that can be summed for the aggregation and normalization process to occur.

Value

A aggregated/normalized user item tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Kmeans: `hai_kmeans_automl_predict()`, `hai_kmeans_automl()`, `hai_kmeans_mapped_tbl()`, `hai_kmeans_obj()`, `hai_kmeans_scree_data_tbl()`, `hai_kmeans_scree_plt()`, `hai_kmeans_tidy_tbl()`
Examples

library(healthyR.data)
library(dplyr)

data_tbl <- healthyR_data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?") %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()

hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
)

hai_knn_data_prepper

Prep Data for k-NN - Recipe

Description

Automatically prep a data.frame/tibble for use in the k-NN algorithm.

Usage

hai_knn_data_prepper(.data, .recipe_formula)

Arguments

.data The data that you are passing to the function. Can be any type of data that is accepted by the data parameter of the recipes::reciep() function.

.recipe_formula The formula that is going to be passed. For example if you are using the iris data then the formula would most likely be something like Species ~ .

Details

This function will automatically prep your data.frame/tibble for use in the k-NN algorithm. The k-NN algorithm is a lazy learning classification algorithm. It expects data to be presented in a certain fashion.

This function will output a recipe specification.

Value

A recipe object
**hai_kurtosis_vec**

**Description**

This function takes in a vector as its input and will return the kurtosis of that vector. The length of this vector must be at least four numbers. The kurtosis explains the sharpness of the peak of a distribution of data.

\[
\frac{\left(\frac{1}{n} \sum (x - \mu)^4\right)}{\left(\frac{1}{n} \sum (x - \mu)^2\right)^2}
\]

**Usage**

`hai_kurtosis_vec(.x)`

**Arguments**

- `.x` A numeric vector of length four or more.

**Details**

A function to return the kurtosis of a vector.

**Value**

The kurtosis of a vector
**hai_polynomial_augment**

**Description**
This function takes in a data table and a predictor column. A user can either create their own formula using the `.formula` parameter or, if they leave the default of NULL then the user must enter a `.degree` AND `.pred_col` column.

**Usage**
```
hai_polynomial_augment(
  .data,
  .formula = NULL,
  .pred_col = NULL,
  .degree = 1,
  .new_col_prefix = "nt_"
)
```

**Arguments**
- `.data` The data being passed that will be augmented by the function.
- `.formula` This should be a valid formula like `y ~ .^2` or NULL.
- `.pred_col` This is passed `rlang::enquo()` to capture the vector that you designate as the 'y' column.
- `.degree` This should be an integer and is used to set the degree in the poly function. The degree must be less than the unique data points or it will error out.
- `.new_col_prefix` The default is "nt_" which stands for "new_term". You can set this to whatever you like, as long as it is a quoted string.
Details

A valid data.frame/tibble must be passed to this function. It is required that a user either enter a .formula or a .degree AND .pred_col otherwise this function will stop and error out.

Under the hood this function will create a \texttt{stats::poly()} function if the .formula is left as NULL. For example:

- .formula = A ~ .^2
- OR .degree = 2, .pred_col = A

There is also a parameter .new_col_prefix which will add a character string to the column names so that they are easily identified further down the line. The default is 'nt_'.

Value

An augmented tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function: \texttt{hai_fourier_augment()}, \texttt{hai_fourier_discrete_augment()}, \texttt{hai_hyperbolic_augment()}, \texttt{hai_scale_zero_one_augment()}, \texttt{hai_scale_zscore_augment()}, \texttt{haiWinsorized_move_augment()}, \texttt{haiWinsorized_truncate_augment()}

Examples

```r
suppressPackageStartupMessages(library(dplyr))
data_tbl <- data.frame(
  A = c(0, 2, 4),
  B = c(1, 3, 5),
  C = c(2, 4, 6)
)

hai_polynomial_augment(.data = data_tbl, .pred_col = A, .degree = 2, .new_col_prefix = "n")
hai_polynomial_augment(.data = data_tbl, .formula = A ~ .^2, .degree = 1)
```

Description

Automatically prep a data.frame/tibble for use in the Ranger algorithm.
Usage

hai_ranger_data_prepper(.data, .recipe_formula)

Arguments

.data The data that you are passing to the function. Can be any type of data that is accepted by the data parameter of the recipes::recipe() function.

.recipe_formula The formula that is going to be passed. For example if you are using the diamonds data then the formula would most likely be something like price ~ .

Details

This function will automatically prep your data.frame/tibble for use in the Ranger algorithm.

This function will output a recipe specification.

Value

A recipe object

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/rand_forest.html

Other Preprocessor: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig(), hai_earth_data_prepper(), hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_svm_poly_data_prepper(), hai_svm_rbf_data_prepper(), hai_xgboost_data_prepper()

Other Ranger: hai_auto_ranger()

Examples

library(ggplot2)

# Regression
hai_ranger_data_prepper(.data = diamonds, .recipe_formula = price ~ .)
reg_obj <- hai_ranger_data_prepper(diamonds, price ~ .)
get_juiced_data(reg_obj)

# Classification
hai_ranger_data_prepper(Titanic, Survived ~ .)
cla_obj <- hai_ranger_data_prepper(Titanic, Survived ~ .)
get_juiced_data(cla_obj)
\textit{hai\_range\_statistic} \hspace{1cm} \textit{Get the range statistic}

\section*{Description}
Takes in a numeric vector and returns back the range of that vector

\section*{Usage}
\texttt{hai\_range\_statistic(.x)}

\section*{Arguments}
\texttt{.x} \hspace{1cm} A numeric vector

\section*{Details}
Takes in a numeric vector and returns the range of that vector using the \texttt{diff} and \texttt{range} functions.

\section*{Value}
A single number, the range statistic

\section*{Author(s)}
Steven P. Sandeson II, MPH

\section*{Examples}
\texttt{hai\_range\_statistic(seq(1:10))}

\newpage
\textit{hai\_scale\_color\_colorblind} \hspace{1cm} \textit{Provide Colorblind Compliant Colors}

\section*{Description}
8 Hex RGB color definitions suitable for charts for colorblind people.

\section*{Usage}
\texttt{hai\_scale\_color\_colorblind(..., theme = "hai")}
Arguments

... Data passed in from a ggplot object
theme Right now this is hai only. Anything else will render an error.

Details

This function is used in others in order to help render plots for those that are color blind.

Value

A ggplot layer

Author(s)

Steven P. Sanderson II, MPH

See Also

Other Color_Blind: color_blind(), hai_scale_fill_colorblind()

Description

8 Hex RGB color definitions suitable for charts for colorblind people.

Usage

hai_scale_fill_colorblind(..., theme = "hai")

Arguments

... Data passed in from a ggplot object
theme Right now this is hai only. Anything else will render an error.

Details

This function is used in others in order to help render plots for those that are color blind.

Value

A ggplot layer

Author(s)

Steven P. Sanderson II, MPH
**hai_scale_zero_one_augment**

*Augment Function Scale Zero One*

**Description**

Takes a numeric vector and will return a vector that has been scaled from [0, 1]

**Usage**

```r
glimpse(hai_scale_zero_one_augment(.data, .value, .names = "auto"))
```

**Arguments**

- `.data` The data being passed that will be augmented by the function.
- `.value` This is passed `rlang::enquo()` to capture the vectors you want to augment.
- `.names` This is set to `’auto’` by default but can be a user supplied character string.

**Details**

Takes a numeric vector and will return a vector that has been scaled from [0, 1].

The input vector must be numeric. The computation is fairly straightforward. This may be helpful when trying to compare the distributions of data where a distribution like beta from the `fitdistrplus` package which requires data to be between 0 and 1

\[ y[h] = (x - \text{min}(x))/(\text{max}(x) - \text{min}(x)) \]

This function is intended to be used on its own in order to add columns to a tibble.

**Value**

An augmented tibble

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Augment Function: `hai_fourier_augment()`, `hai_fourier_discrete_augment()`, `hai_hyperbolic_augment()`, `hai_polynomial_augment()`, `hai_scale_zscore_augment()`, `hai_winsorized_move_augment()`, `hai_winsorized_truncate_augment()`

Other Scale: `hai_scale_zero_one_vec()`, `hai_scale_zscore_augment()`, `hai_scale_zscore_vec()`, `step_hai_scale_zscore()`
**Examples**

```r
df <- data.frame(x = rnorm(100, 2, 1))
hai_scale_zero_one_augment(df, x)
```

---

### Vector Function Scale to Zero and One

**Description**

Takes a numeric vector and will return a vector that has been scaled from $[0, 1]$

**Usage**

```r
hai_scale_zero_one_vec(.x)
```

**Arguments**

- `.x` A numeric vector to be scaled from $[0, 1]$ inclusive.

**Details**

Takes a numeric vector and will return a vector that has been scaled from $[0, 1]$ The input vector must be numeric. The computation is fairly straightforward. This may be helpful when trying to compare the distributions of data where a distribution like beta from the `fitdistrplus` package which requires data to be between 0 and 1

$$y[h] = (x - \text{min}(x))/(\text{max}(x) - \text{min}(x))$$

This function can be used on it’s own. It is also the basis for the function `hai_scale_zero_one_augment()`.

**Value**

A numeric vector

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**

Other Vector Function: `hai_fourier_discrete_vec()`, `hai_fourier_vec()`, `hai_hyperbolic_vec()`, `hai_kurtosis_vec()`, `hai_scale_zscore_vec()`, `hai_skewness_vec()`, `hai_winsorized_move_vec()`, `hai_winsorized_truncate_vec()`

Other Scale: `hai_scale_zero_one_augment()`, `hai_scale_zscore_augment()`, `hai_scale_zscore_vec()`, `step_hai_scale_zscore()`
Examples

```r
vec_1 <- rnorm(100, 2, 1)
vec_2 <- hai_scale_zero_one_vec(vec_1)

dens_1 <- density(vec_1)
dens_2 <- density(vec_2)
max_x <- max(dens_1$x, dens_2$x)
max_y <- max(dens_1$y, dens_2$y)
plot(dens_1,
    asp = max_y / max_x, main = "Density vec_1 (Red) and vec_2 (Blue)",
    col = "red", xlab = "", ylab = "Density of Vec 1 and Vec 2"
)
lines(dens_2, col = "blue")
```

hai_scale_zscore_augment

Augment Function Scale Zero One

Description

Takes a numeric vector and will return a vector that has been scaled by mean and standard deviation.

Usage

```r
hai_scale_zscore_augment(.data, .value, .names = "auto")
```

Arguments

- `.data` The data being passed that will be augmented by the function.
- `.value` This is passed `rlang::enquo()` to capture the vectors you want to augment.
- `.names` This is set to 'auto' by default but can be a user supplied character string.

Details

Takes a numeric vector and will return a vector that has been scaled by mean and standard deviation. The input vector must be numeric. The computation is fairly straightforward. This may be helpful when trying to compare the distributions of data where a distribution like beta from the `fitdistrplus` package which requires data to be between 0 and 1

\[ y[h] = \frac{x - \text{mean}(x)}{\text{sd}(x)} \]

This function is intended to be used on its own in order to add columns to a tibble.

Value

An augmented tibble
Author(s)

Steven P. Sanderson II, MPH

See Also

Other Augment Function: `hai_fourier_augment()`, `hai_fourier_discrete_augment()`, `hai_hyperbolic_augment()`, `hai_polynomial_augment()`, `hai_scale_zero_one_augment()`, `hai_winsorized_move_augment()`, `hai_winsorized_truncate_augment()`

Other Scale: `hai_scale_zero_one_augment()`, `hai_scale_zero_one_vec()`, `hai_scale_zscore_vec()`, `step_hai_scale_zscore()`

Examples

```r
df <- data.frame(x = mtcars$mpg)
hai_scale_zscore_augment(df, x)
```

---

**hai_scale_zscore_vec**  
*Vector Function Scale to Zero and One*

Description

Takes a numeric vector and will return a vector that has been scaled from mean and standard deviation.

Usage

```r
hai_scale_zscore_vec(.x)
```

Arguments

`.x`  
A numeric vector to be scaled by mean and standard deviation inclusive.

Details

Takes a numeric vector and will return a vector that has been scaled from mean and standard deviation.

The input vector must be numeric. The computation is fairly straightforward. This may be helpful when trying to compare the distributions of data where a distribution like beta from the `fitdistrplus` package which requires data to be between 0 and 1

\[
y[h] = (x - \text{mean}(x)/\text{sd}(x))
\]

This function can be used on its own. It is also the basis for the function `hai_scale_zscore_augment()`.

Value

A numeric vector
**hai_skewed_features**

*Get Skewed Feature Columns*

**Description**

Takes in a data.frame/tibble and returns a vector of names of the columns that are skewed.

**Usage**

```r
hai_skewed_features(.data, .threshold = 0.6, .drop_keys = NULL)
```

**Arguments**

- `.data` The data.frame/tibble you are passing in.
- `.threshold` A level of skewness that indicates where you feel a column should be considered skewed.
- `.drop_keys` A c() character vector of columns you do not want passed to the function.

**Details**

Takes in a data.frame/tibble and returns a vector of names of the skewed columns. There are two other parameters. The first is the `.threshold` parameter that is set to the level of skewness you want in order to consider the column too skewed. The second is `.drop_keys`, these are columns you don’t want to be considered for whatever reason in the skewness calculation.

---

**Examples**

```r
cvec_1 <- mtcars$mpg
cvec_2 <- hai_scale_zscore_vec(vec_1)

ax <- pretty(min(vec_1, vec_2):max(vec_1, vec_2), n = 12)

hist(vec_1, breaks = ax, col = "blue")
hist(vec_2, breaks = ax, col = "red", add = TRUE)
```
Value

A character vector of column names that are skewed.

Author(s)

Steven P. Sandeson II, MPH

Examples

hai_skewed_features(mtcars)
hai_skewed_features(mtcars, .drop_keys = c("mpg", "hp"))
hai_skewed_features(mtcars, .drop_keys = "hp")

hai_skewness_vec

**Compute Skewness of a Vector**

Description

This function takes in a vector as it’s input and will return the skewness of that vector. The length of this vector must be at least four numbers. The skewness explains the ‘tailedness’ of the distribution of data.

\[
\frac{(1/n) \cdot \sum(x - \mu)^3}{((1/n) \cdot \sum(x - \mu)^2)^{3/2}}
\]

Usage

hai_skewness_vec(.x)

Arguments

.x A numeric vector of length four or more.

Details

A function to return the skewness of a vector.

Value

The skewness of a vector

Author(s)

Steven P. Sanderson II, MPH
**hai_svm_poly_data_prepper**

**See Also**


Other Vector Function: `hai_fourier_discrete_vec()` , `hai_fourier_vec()` , `hai_hyperbolic_vec()` , `hai_kurtosis_vec()` , `hai_scale_zero_one_vec()` , `hai_scale_zscore_vec()` , `hai_winsorized_move_vec()` , `hai_winsorized_truncate_vec()`

**Examples**

```r
hai_skewness_vec(rnorm(100, 3, 2))
```

---

**hai_svm_poly_data_prepper**

*Prep Data for SVM_Poly - Recipe*

---

**Description**

Automatically prep a data.frame/tibble for use in the SVM_Poly algorithm.

**Usage**

`hai_svm_poly_data_prepper(.data, .recipe_formula)`

**Arguments**

- `.data` The data that you are passing to the function. Can be any type of data that is accepted by the `data` parameter of the `recipes::recipe()` function.
- `.recipe_formula` The formula that is going to be passed. For example if you are using the diamonds data then the formula would most likely be something like `price ~ .`.

**Details**

This function will automatically prep your data.frame/tibble for use in the SVM_Poly algorithm. The SVM_Poly algorithm is for regression only. This function will output a recipe specification.

**Value**

A recipe object

**Author(s)**

Steven P. Sanderson II, MPH
hai_svm_rbf_data_prepper

See Also

https://parsnip.tidymodels.org/reference/svm_poly.html

Other Preprocessors: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_impute(),
hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig(), hai_earth_data_prepper(),
hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_rbf_data_prepper(),
hai_xgboost_data_prepper()

Other SVM_Poly: hai_auto_svm_poly()

Examples

library(ggplot2)

# Regression
hai_svm_poly_data_prepper(.data = diamonds, .recipe_formula = price ~ .)
reg_obj <- hai_svm_poly_data_prepper(diamonds, price ~ .)
get_juiced_data(reg_obj)

# Classification
hai_svm_poly_data_prepper(Titanic, Survived ~ .)
cla_obj <- hai_svm_poly_data_prepper(Titanic, Survived ~ .)
get_juiced_data(cla_obj)

hai_svm_rbf_data_prepper

Prep Data for SVM_RBF - Recipe

Description

Automatically prep a data.frame/tibble for use in the SVM_RBF algorithm.

Usage

hai_svm_rbf_data_prepper(.data, .recipe_formula)

Arguments

.data The data that you are passing to the function. Can be any type of data that is
accepted by the data parameter of the recipes::recipe() function.

.recipe_formula The formula that is going to be passed. For example if you are using the
diamonds data then the formula would most likely be something like price ~ .
Details

This function will automatically prep your data.frame/tibble for use in the SVM_RBF algorithm. The SVM_RBF algorithm is for regression only.
This function will output a recipe specification.

Value

A recipe object

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/svm_rbf.html

Other Preprocessor: hai_c50_data_prepper(), hai_cubist_data_prepper(), hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig(), hai_earth_data_prepper(), hai_glmnet_data_prepper(), hai_knn_data_prepper(), hai_ranger_data_prepper(), hai_svm_poly_data_prepper(), hai_xgboost_data_prepper()

Other SVM_RBF: hai_auto_svm_rbf()

Examples

library(ggplot2)

# Regression
hai_svm_rbf_data_prepper(.data = diamonds, .recipe_formula = price ~ .)
reg_obj <- hai_svm_rbf_data_prepper(diamonds, price ~ .)
get_juiced_data(reg_obj)

# Classification
hai_svm_rbf_data_prepper(Titanic, Survived ~ .)
cla_obj <- hai_svm_rbf_data_prepper(Titanic, Survived ~ .)
get_juiced_data(cla_obj)

hai_umap_list

Description

Create a umap object from the uwot::umap() function.

Usage

hai_umap_list(.data, .kmeans_map_tbl, .k_cluster = 5)

umap_list(.data, .kmeans_map_tbl, .k_cluster = 5)
Arguments

- `.data` The data from the `hai_kmeans_user_item_tbl()` function.
- `.kmeans_map_tbl` The data from the `hai_kmeans_mapped_tbl()`.
- `.k_cluster` Pick the desired amount of clusters from your analysis of the scree plot.

Details

This takes in the user item table/matix that is produced by `hai_kmeans_user_item_tbl()` function. This function uses the defaults of `uwot::umap()`.

Value

A list of tibbles and the umap object

Author(s)

Steven P. Sanderson II, MPH

See Also

- [https://cran.r-project.org/package=uwot](https://cran.r-project.org/package=uwot) (CRAN)
- [https://github.com/jlmelville/uwot](https://github.com/jlmelville/uwot) (GitHub)
- [https://github.com/jlmelville/uwot](https://github.com/jlmelville/uwot) (arXiv paper)

Other UMAP: `hai_umap_plot()`

Examples

```r
library(healthyR.data)
library(dplyr)
library(broom)

data_tbl <- healthyR_data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?") %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()

uit_tbl <- hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
)

kmm_tbl <- hai_kmeans_mapped_tbl(uit_tbl)
```
Description

Create a UMAP Projection plot.

Usage

```
hai_umap_plot(.data, .point_size = 2, .label = TRUE)
```

Arguments

- `.data` The data from the `umap_list()` function.
- `.point_size` The desired size for the points of the plot.
- `.label` Should `ggrepel::geom_label_repel()` be used to display cluster user labels.

Details

This takes in `umap_kmeans_cluster_results_tbl` from the `umap_list()` function output.

Value

A ggplot2 UMAP Projection with clusters represented by colors.

Author(s)

Steven P. Sanderson II, MPH

See Also

- [https://cran.r-project.org/package=uwot](https://cran.r-project.org/package=uwot) (CRAN)
- [https://github.com/jlmelville/uwot](https://github.com/jlmelville/uwot) (GitHub)
- [https://github.com/jlmelville/uwot](https://github.com/jlmelville/uwot) (arXiv paper)

Other UMAP: `hai_umap_list()`
Examples

```r
library(healthyR.data)
library(dplyr)
library(broom)
library(ggplot2)

data_tbl <- healthyR_data %>%
  filter(ip_op_flag == "I") %>%
  filter(payer_grouping != "Medicare B") %>%
  filter(payer_grouping != "?") %>%
  select(service_line, payer_grouping) %>%
  mutate(record = 1) %>%
  as_tibble()

uit_tbl <- hai_kmeans_user_item_tbl(
  .data = data_tbl,
  .row_input = service_line,
  .col_input = payer_grouping,
  .record_input = record
)

kmm_tbl <- hai_kmeans_mapped_tbl(uit_tbl)

ump_lst <- hai_umap_list(.data = uit_tbl, kmm_tbl, 3)

hai_umap_plot(.data = ump_lst, .point_size = 3)
```

---

**hai_winsorized_move_augment**

*Augment Function Winsorize Move*

**Description**

Takes a numeric vector and will return a tibble with the winsorized values.

**Usage**

```r
hai_winsorized_move_augment(.data, .value, .multiple, .names = "auto")
```

**Arguments**

- `.data` - The data being passed that will be augmented by the function.
- `.value` - This is passed `rlang::enquo()` to capture the vectors you want to augment.
- `.multiple` - A positive number indicating how many times the the zero center mean absolute deviation should be multiplied by for the scaling parameter.
- `.names` - The default is "auto"
Details

Takes a numeric vector and will return a winsorized vector of values that have been moved some multiple from the mean absolute deviation zero center of some vector. The intent of winsorization is to limit the effect of extreme values.

Value

An augmented tibble

Author(s)

Steven P. Sanderson II, MPH

See Also

https://en.wikipedia.org/wiki/Winsorizing

Other Augment Function: hai_fourier_augment(), hai_fourier_discrete_augment(), hai_hyperbolic_augment(), hai_polynomial_augment(), hai_scale_zero_one_augment(), hai_scale_zscore_augment(), hai_winsorized_truncate_augment()

Examples

suppressPackageStartupMessages(library(dplyr))

len_out <- 24
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
  )

hai_winsorized_move_augment(data_tbl, a, .multiple = 3)
**Arguments**

- `.x`  A numeric vector
- `.multiple`  A positive number indicating how many times the the zero center mean absolute deviation should be multiplied by for the scaling parameter.

**Details**

Takes a numeric vector and will return a winsorized vector of values that have been moved some multiple from the mean absolute deviation zero center of some vector. The intent of winsorization is to limit the effect of extreme values.

**Value**

A numeric vector

**Author(s)**

Steven P. Sanderson II, MPH

**See Also**


This function can be used on its own. It is also the basis for the function `hai_winsorized_move_augment()`. Other Vector Function: `hai_fourier_discrete_vec()`, `hai_fourier_vec()`, `hai_hyperbolic_vec()`, `hai_kurtosis_vec()`, `hai_scale_zero_one_vec()`, `hai_scale_zscore_vec()`, `hai_skewness_vec()`, `hai_winsorized_truncate_vec()`

**Examples**

```r
suppressPackageStartupMessages(library(dplyr))

len_out <- 25
by_unit <- "month"
start_date <- as.Date("2021-01-01")
data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

vec_1 <- hai_winsorized_move_vec(data_tbl$a, .multiple = 1)

plot(data_tbl$a)
lines(data_tbl$a)
lines(vec_1, col = "blue")
```
Description
Takes a numeric vector and will return a tibble with the winsorized values.

Usage
hai_winsorized_truncate_augment(.data, .value, .fraction, .names = "auto")

Arguments
.data The data being passed that will be augmented by the function.
.value This is passed rlang::enquo() to capture the vectors you want to augment.
.fraction A positive fractional between 0 and 0.5 that is passed to the stats::quantile parameter of probs.
.names The default is "auto"

Details
Takes a numeric vector and will return a winsorized vector of values that have been truncated if they are less than or greater than some defined fraction of a quantile. The intent of winsorization is to limit the effect of extreme values.

Value
An augmented tibble

Author(s)
Steven P. Sanderson II, MPH

See Also
https://en.wikipedia.org/wiki/Winsorizing
Other Augment Function: hai_fourier_augment(), hai_fourier_discrete_augment(), hai_hyperbolic_augment(), hai_polynomial_augment(), hai_scale_zero_one_augment(), hai_scale_zscore_augment(), hai_winsorized_move_augment()
Examples

```r
suppressPackageStartupMessages(library(dplyr))

len_out <- 24
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

hai_winsorized_truncate_augment(data_tbl, a, .fraction = 0.05)
```

---

**hai_winsorized_truncate_vec**

*Vector Function Winsorize Truncate*

### Description

Takes a numeric vector and will return a vector of winsorized values.

### Usage

```r
hai_winsorized_truncate_vec(.x, .fraction = 0.05)
```

### Arguments

- **.x**
  - A numeric vector

- **.fraction**
  - A positive fractional between 0 and 0.5 that is passed to the `stats::quantile` parameter of `probs`.

### Details

Takes a numeric vector and will return a winsorized vector of values that have been truncated if they are less than or greater than some defined fraction of a quantile. The intent of winsorization is to limit the effect of extreme values.

### Value

A numeric vector

### Author(s)

Steven P. Sanderson II, MPH
See Also

https://en.wikipedia.org/wiki/Winsorizing

This function can be used on its own. It is also the basis for the function `hai_winsorized_truncate_augment()`.

Other Vector Function: `hai_fourier_discrete_vec()`, `hai_fourier_vec()`, `hai_hyperbolic_vec()`, `hai_kurtosis_vec()`, `hai_scale_zero_one_vec()`, `hai_scale_zscore_vec()`, `hai_skewness_vec()`, `hai_winsorized_move_vec()`

Examples

```r
suppressPackageStartupMessages(library(dplyr))

len_out <- 25
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

vec_1 <- hai_winsorized_truncate_vec(data_tbl$a, .fraction = 0.05)

plot(data_tbl$a)
lines(data_tbl$a)
lines(vec_1, col = "blue")
```

Prep Data for XGBoost - Recipe

**Description**

Automatically prep a data.frame/tibble for use in the xgboost algorithm.

**Usage**

```r
hai_xgboost_data_prepper(.data, .recipe_formula)
```

**Arguments**

- `.data` The data that you are passing to the function. Can be any type of data that is accepted by the `data` parameter of the `recipes::recipe()` function.
- `.recipe_formula` The formula that is going to be passed. For example if you are using the diamonds data then the formula would most likely be something like `price ~ .`.  

Details

This function will automatically prep your data.frame/tibble for use in the XGBoost algorithm. This function will output a recipe specification.

Value

A recipe object

Author(s)

Steven P. Sanderson II, MPH

See Also

https://parsnip.tidymodels.org/reference/details_boost_tree_xgboost.html

Other Preprocessor: 
- hai_c50_data_prepper()
- hai_cubist_data_prepper()
- hai_data_impute()
- hai_data_poly()
- hai_data_scale()
- hai_data_transform()
- hai_data_trig()
- hai_earth_data_prepper()
- hai_glmnet_data_prepper()
- hai_knn_data_prepper()
- hai_ranger_data_prepper()
- hai_svm_poly_data_prepper()
- hai_svm_rbf_data_prepper()

Examples

library(ggplot2)

# Regression
hai_xgboost_data_prepper(.data = diamonds, .recipe_formula = price ~ .)
reg_obj <- hai_xgboost_data_prepper(diamonds, price ~ .)
get_juiced_data(reg_obj)

# Classification
hai_xgboost_data_prepper(Titanic, Survived ~ .)
cla_obj <- hai_xgboost_data_prepper(Titanic, Survived ~ .)
get_juiced_data(cla_obj)

pca_your_recipe

Perform PCA

Description

This is a simple function that will perform PCA analysis on a passed recipe.

Usage

pca_your_recipe(.recipe_object, .data, .threshold = 0.75, .top_n = 5)
Arguments

.recipe_object  The recipe object you want to pass.
data  The full data set that is used in the original recipe object passed into .recipe_object in order to obtain the baked data of the transform.
.threshold  A number between 0 and 1. A fraction of the total variance that should be covered by the components.
top_n  How many variables loadings should be returned per PC

Details

This is a simple wrapper around some recipes functions to perform a PCA on a given recipe. This function will output a list and return it invisible. All of the components of the analysis will be returned in a list as their own object that can be selected individually. A scree plot is also included. The items that get returned are:

1. pca_transform - This is the pca recipe.
2. variable_loadings
3. variable_variance
4. pca_estimates
5. pca_juiced_estimates
6. pca_baked_data
7. pca_variance_df
8. pca_rotation_df
9. pca_variance_scree_plt
10. pca_loadings_plt
11. pca_loadings_plotly
12. pca_top_n_loadings_plt
13. pca_top_n_plotly

Value

A list object with several components.

Author(s)

Steven P. Sanderson II, MPH

See Also

https://recipes.tidymodels.org/reference/step_pca.html
Other Data Wrangling: get_juiced_data()
Other Data Recipes: hai_data_impute(), hai_data_poly(), hai_data_scale(), hai_data_transform(), hai_data_trig()
Examples

```r
suppressPackageStartupMessages(library(timetk))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(purrr))
suppressPackageStartupMessages(library(healthyR.data))
suppressPackageStartupMessages(library(rsample))
suppressPackageStartupMessages(library(recipes))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(plotly))

data_tbl <- healthyR_data %>%
  select(visit_end_date_time) %>%
  summarise_by_time(
    .date_var = visit_end_date_time,
    .by = "month",
    value = n()
  ) %>%
  set_names("date_col", "value") %>%
  filter_by_time(
    .date_var = date_col,
    .start_date = "2013",
    .end_date = "2020"
  ) %>%
  mutate(date_col = as.Date(date_col))

splits <- initial_split(data = data_tbl, prop = 0.8)

rec_obj <- recipe(value ~ ., training(splits)) %>%
  step_timeseries_signature(date_col) %>%
  step_rm(matches("(iso$)|(xts$)|(hour)|(min)|(sec)|(am.pm)$"))

output_list <- pca_your_recipe(rec_obj, .data = data_tbl)
output_list$pca_variance_scree_plt
output_list$pca_loadings_plt
output_list$pca_top_n_loadings_plt
```

---

**step_hai_fourier**

**Recipes Step Fourier Generator**

**Description**

*step_hai_fourier* creates a a **specification** of a recipe step that will convert numeric data into either a 'sin', 'cos', or 'sincos' feature that can aid in machine learning.

**Usage**

```r
step_hai_fourier(
  recipe,
```
...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  scale_type = c("sin", "cos", "sincos"),
  period = 1,
  order = 1,
  skip = FALSE,
  id = rand_id("hai_fourier")
)

Arguments

recipe A recipe object. The step will be added to the sequence of operations for this recipe.

... One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric

role For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.

trained A logical to indicate if the quantities for preprocessing have been estimated.

columns A character string of variables that will be used as inputs. This field is a placeholder and will be populated once recipes::prep() is used.

scale_type A character string of a scaling type, one of "sin", "cos", or "sincos"

period The number of observations that complete a cycle

order The fourier term order

skip A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

Details

Numeric Variables Unlike other steps, step_hai_fourier does not remove the original numeric variables. recipes::step_rm() can be used for this purpose.

Value

For step_hai_fourier, an updated version of recipe with the new step added to the sequence of existing steps (if any).

Main Recipe Functions:

• recipes::recipe()
• recipes::prep()
• recipes::bake()
See Also

Other Recipes: `step_hai_fourier_discrete()`, `step_hai_hyperbolic()`, `step_hai_scale_zero_one()`, `step_hai_scale_zscore()`, `step_haiWinsorized_move()`, `step_haiWinsorized_truncate()

Examples

```r
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

len_out <- 10
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

# Create a recipe object
rec_obj <- recipe(a ~ ., data = data_tbl) %>%
  step_hai_fourier(b, scale_type = "sin") %>%
  step_hai_fourier(b, scale_type = "cos") %>%
  step_hai_fourier(b, scale_type = "sincos")

# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>% get_juiced_data()
```

---

`step_hai_fourier_discrete`

*Recipes Step Fourier Discrete Generator*

**Description**

`step_hai_fourier_discrete` creates a specification of a recipe step that will convert numeric data into either a 'sin', 'cos', or 'sincos' feature that can aid in machine learning.
Usage

```r
step_hai_fourier_discrete(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  scale_type = c("sin", "cos", "sincos"),
  period = 1,
  order = 1,
  skip = FALSE,
  id = rand_id("hai_fourier_discrete")
)
```

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric or date.POSIXct.
- **role**: For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **columns**: A character string of variables that will be used as inputs. This field is a placeholder and will be populated once `recipes::prep()` is used.
- **scale_type**: A character string of a scaling type, one of "sin", "cos", or "sincos".
- **period**: The number of observations that complete a cycle.
- **order**: The fourier term order.
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.

Details

**Numeric Variables** Unlike other steps, `step_hai_fourier_discrete` does not remove the original numeric variables. `recipes::step_rm()` can be used for this purpose.

Value

For `step_hai_fourier_discrete`, an updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Main Recipe Functions:


• recipes::recipe()
• recipes::prep()
• recipes::bake()

See Also

Other Recipes: step_hai_fourier(), step_hai_hyperbolic(), step_hai_scale_zero_one(), step_hai_scale_zscore(), step_hai_winsorized_move(), step_hai_winsorized_truncate()

Examples

suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

len_out <- 10
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

# Create a recipe object
rec_obj <- recipe(a ~ ., data = data_tbl) %>%
  step_hai_fourier_discrete(b, scale_type = "sin") %>%
  step_hai_fourier_discrete(b, scale_type = "cos") %>%
  step_hai_fourier_discrete(b, scale_type = "sincos")

# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>% get_juiced_data()

---

step_hai_hyperbolic  Recipes Step Hyperbolic Generator

Description

step_hai_hyperbolic creates a specification of a recipe step that will convert numeric data into either a ‘sin’, ‘cos’, or ‘tan’ feature that can aid in machine learning.
Usage

```r
step_hai_hyperbolic(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  scale_type = c("sin", "cos", "tan", "sincos"),
  skip = FALSE,
  id = rand_id("hai_hyperbolic")
)
```

Arguments

- **recipe**: A recipe object. The step will be added to the sequence of operations for this recipe.
- **...**: One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class `numeric`.
- **role**: For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
- **trained**: A logical to indicate if the quantities for preprocessing have been estimated.
- **columns**: A character string of variables that will be used as inputs. This field is a placeholder and will be populated once `recipes::prep()` is used.
- **scale_type**: A character string of a scaling type, one of "sin", "cos", "tan" or "sincos".
- **skip**: A logical. Should the step be skipped when the recipe is baked by `bake.recipe()`? While all operations are baked when `prep.recipe()` is run, some operations may not be able to be conducted on new data (e.g., processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- **id**: A character string that is unique to this step to identify it.

Details

**Numeric Variables** Unlike other steps, `step_hai_hyperbolic` does not remove the original numeric variables. `recipes::step_rm()` can be used for this purpose.

Value

For `step_hai_hyperbolic`, an updated version of `recipe` with the new step added to the sequence of existing steps (if any).

Main Recipe Functions:

- `recipes::recipe()`
- `recipes::prep()`
- `recipes::bake()`
See Also

Other Recipes: `step_hai_fourier_discrete()`, `step_hai_fourier()`, `step_hai_scale_zero_one()`, `step_hai_scale_zscore()`, `step_hai_winsorized_move()`, `step_hai_winsorized_truncate()`

Examples

```r
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

len_out <- 10
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

# Create a recipe object
deepstep obj <- recipe(a ~ ., data = data_tbl) %>%
  step_hai_hyperbolic(b, scale_type = "sin") %>%
  step_hai_hyperbolic(b, scale_type = "cos")

# View the recipe object
deepstep obj

# Prepare the recipe object
prep(deepstep obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(deepstep obj), data_tbl)

deepstep obj %>% get_juiced_data()
```

---

**step_hai_scale_zero_one**

*Recipes Data Scale to Zero and One*

**Description**

`step_hai_scale_zero_one` creates a *specification* of a recipe step that will convert numeric data into from a time series into its velocity.

**Usage**

```r
step_hai_scale_zero_one(
  recipe,  
```
...,
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("hai_scale_zero_one")
)

Arguments

recipe A recipe object. The step will be added to the sequence of operations for this recipe.
...
One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric
role For model terms created by this step, what analysis role should they be assigned? By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
trained A logical to indicate if the quantities for preprocessing have been estimated.
columns A character string of variables that will be used as inputs. This field is a placeholder and will be populated once recipes::prep() is used.
skip A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id A character string that is unique to this step to identify it.

Details

Numeric Variables Unlike other steps, step_hai_scale_zero_one does not remove the original numeric variables. recipes::step_rm() can be used for this purpose.

Value

For step_hai_scale_zero_one, an updated version of recipe with the new step added to the sequence of existing steps (if any).

Main Recipe Functions:

• recipes::recipe()
• recipes::prep()
• recipes::bake()

See Also

Other Recipes: step_hai_fourier_discrete(), step_hai_fourier(), step_hai_hyperbolic(), step_hai_scale_zscore(), step_haiWinsorized_move(), step_haiWinsorized_truncate()
Examples

```r
 suppressPackageStartupMessages(library(dplyr))
 suppressPackageStartupMessages(library(recipes))

data_tbl <- data.frame(a = rnorm(200, 3, 1), b = rnorm(200, 2, 2))

# Create a recipe object
rec_obj <- recipe(a ~ ., data = data_tbl) %>%
  step_hai_scale_zero_one(b)

# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>%
  prep() %>%
  juice()
```

---

### step_hai_scale_zscore  Recipes Data Scale by Z-Score

**Description**

`step_hai_scale_zscore` creates a specification of a recipe step that will convert numeric data into from a time series into its velocity.

**Usage**

```r
step_hai_scale_zscore(
  recipe,
  ...,  # other arguments
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  skip = FALSE,
  id = rand_id("hai_scale_zscore")
)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
... One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric

role For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.

trained A logical to indicate if the quantities for preprocessing have been estimated.
columns A character string of variables that will be used as inputs. This field is a placeholder and will be populated once recipes::prep() is used.
skip A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
id A character string that is unique to this step to identify it.

Details

Numeric Variables Unlike other steps, step_hai_scale_zscore does not remove the original numeric variables. recipes::step_rm() can be used for this purpose.

Value

For step_hai_scale_zscore, an updated version of recipe with the new step added to the sequence of existing steps (if any).

Main Recipe Functions:

- recipes::recipe()
- recipes::prep()
- recipes::bake()

See Also

Other Recipes: step_hai_fourier_discrete(), step_hai_fourier(), step_hai_hyperbolic(), step_hai_scale_zero_one(), step_hai_winsorized_move(), step_hai_winsorized_truncate()

Other Scale: hai_scale_zero_one_augment(), hai_scale_zero_one_vec(), hai_scale_zscore_augment(), hai_scale_zscore_vec()

Examples

suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

data_tbl <- data.frame(
a = mtcars$mpg,
b = AirPassengers %>% as.vector() %>% head(32)
)

# Create a recipe object
```
rec_obj <- recipe(a ~ ., data = data_tbl) %>%
  step_hai_winsorized_move(b)

# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>%
  prep() %>%
juice()
```

---

**step_hai_winsorized_move**  
*Recipes Step Winsorized Move Generator*

---

**Description**

`step_hai_winsorized_move` creates a specification of a recipe step that will winsorize numeric data.

**Usage**

```
step_hai_winsorized_move(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  multiple = 3,
  skip = FALSE,
  id = rand_id("hai_winsorized_move")
)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric.
- `role` For model terms created by this step, what analysis role should they be assigned to? By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
step_hai Winsorized Move

**trained**
A logical to indicate if the quantities for preprocessing have been estimated.

**columns**
A character string of variables that will be used as inputs. This field is a placeholder and will be populated once recipes::prep() is used.

**multiple**
A positive number indicating how many times the zero center mean absolute deviation should be multiplied by for the scaling parameter.

**skip**
A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

**id**
A character string that is unique to this step to identify it.

**Details**

**Numeric Variables**
Unlike other steps, step_hai Winsorize_move does not remove the original numeric variables. recipes::step_rm() can be used for this purpose.

**Value**
For step_hai Winsorize_move, an updated version of recipe with the new step added to the sequence of existing steps (if any).

**Main Recipe Functions:**
- recipes::recipe()
- recipes::prep()
- recipes::bake()

**See Also**
Other Recipes: step_hai_fourier_discrete(), step_hai_fourier(), step_hai_hyperbolic(), step_hai_scale_zero_one(), step_hai_scale_zscore(), step_hai Winsorized_truncate()

**Examples**
```
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

len_out <- 10
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

# Create a recipe object
rec_obj <- recipe(b ~ ., data = data_tbl) %>%
```
step_hai_winsorized_truncate(a, multiple = 3)

# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>% get_juiced_data()

---

step_hai_winsorized_truncate

*Recipes Step Winsorized Truncate Generator*

**Description**

`step_hai_winsorized_truncate` creates a specification of a recipe step that will winsorize numeric data.

**Usage**

```r
step_hai_winsorized_truncate(
  recipe,
  ..., 
  role = "predictor",
  trained = FALSE,
  columns = NULL,
  fraction = 0.05,
  skip = FALSE,
  id = rand_id("hai_winsorized_truncate")
)
```

**Arguments**

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose which variables that will be used to create the new variables. The selected variables should have class numeric.
- `role` For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new variable columns created by the original variables will be used as predictors in a model.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
**step_hai_winsorized_truncate**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columns</td>
<td>A character string of variables that will be used as inputs. This field is a placeholder and will be populated once <code>recipes::prep()</code> is used.</td>
</tr>
<tr>
<td>fraction</td>
<td>A positive fractional between 0 and 0.5 that is passed to the <code>stats::quantile</code> parameter of <code>probs</code>.</td>
</tr>
<tr>
<td>skip</td>
<td>A logical. Should the step be skipped when the recipe is baked by <code>bake.recipe()</code>? While all operations are baked when <code>prep.recipe()</code> is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using <code>skip = TRUE</code> as it may affect the computations for subsequent operations.</td>
</tr>
<tr>
<td>id</td>
<td>A character string that is unique to this step to identify it.</td>
</tr>
</tbody>
</table>

**Details**

**Numeric Variables** Unlike other steps, `step_hai_winsorize_truncate` does *not* remove the original numeric variables. `recipes::step_rm()` can be used for this purpose.

**Value**

For `step_hai_winsorize_truncate`, an updated version of recipe with the new step added to the sequence of existing steps (if any).

**Main Recipe Functions:**

- `recipes::recipe()`
- `recipes::prep()`
- `recipes::bake()`

**See Also**

Other Recipes: `step_hai_fourier_discrete()`, `step_hai_fourier()`, `step_hai_hyperbolic()`, `step_hai_scale_zero_one()`, `step_hai_scale_zscore()`, `step_hai_winsorized_move()`

**Examples**

```r
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(recipes))

len_out <- 10
by_unit <- "month"
start_date <- as.Date("2021-01-01")

data_tbl <- tibble(
  date_col = seq.Date(from = start_date, length.out = len_out, by = by_unit),
  a = rnorm(len_out),
  b = runif(len_out)
)

# Create a recipe object
rec_obj <- recipe(b ~ ., data = data_tbl) %>%
  step_hai_winsorized_truncate(a, fraction = 0.05)
```
# View the recipe object
rec_obj

# Prepare the recipe object
prep(rec_obj)

# Bake the recipe object - Adds the Time Series Signature
bake(prep(rec_obj), data_tbl)

rec_obj %>% get_juiced_data()
Index

* **Augment Function**
  - hai_fourier_augment, 45
  - hai_fourier_discrete_augment, 47
  - hai_hyperbolic_augment, 56
  - hai_polynomial_augment, 72
  - hai_scale_zero_one_augment, 77
  - hai_scale_zscore_augment, 79
  - haiWinsorized_move_augment, 88
  - haiWinsorized_truncate_augment, 91

* **Boiler_Plate**
  - hai_auto_c50, 5
  - hai_auto_cubist, 7
  - hai_auto_earth, 9
  - hai_auto_glmnet, 10
  - hai_auto_knn, 12
  - hai_auto_ranger, 14
  - hai_auto_svm_poly, 15
  - hai_auto_svm_rbf, 17
  - hai_auto_wflw_metrics, 19
  - hai_auto_xgboost, 20

* **C5.0**
  - hai_auto_c50, 5
  - hai_c50_data_prepper, 22

* **Color_Blind**
  - color_blind, 3
  - hai_scale_color_colorblind, 75
  - hai_scale_fill_colorblind, 76

* **Control Charts**
  - hai_control_chart, 23

* **Data Recipes**
  - hai_data_impute, 26
  - hai_data_poly, 28
  - hai_data_scale, 30
  - hai_data_transform, 32
  - hai_data_trig, 35
  - pca_your_recipe, 94

* **Data Wrangling**
  - get_juiced_data, 4
  - pca_your_recipe, 94

* **Default Metric Sets**
  - hai_default_classification_metric_set, 37
  - hai_default_regression_metric_set, 37

* **Dimension Reduction**
  - pca_your_recipe, 94

* **Distribution Functions**
  - hai_distribution_comparison_tbl, 42
  - hai_get_density_data_tbl, 52
  - hai_get_dist_data_tbl, 53

* **Distribution Plots**
  - hai_density_hist_plot, 38
  - hai_density_plot, 39
  - hai_density_qq_plot, 41

* **Earth**
  - hai_auto_earth, 9
  - hai_earth_data_prepper, 44

* **Kmeans**
  - hai_kmeans_automl, 59
  - hai_kmeans_automl_predict, 61
  - hai_kmeans_mapped_tbl, 62
  - hai_kmeans_obj, 63
  - hai_kmeans_scree_data_tbl, 65
  - hai_kmeans_scree_plt, 66
  - hai_kmeans_tidy_tbl, 67
  - hai_kmeans_user_item_tbl, 69

* **Metric_Collection**
  - hai_auto_wflw_metrics, 19

* **Preprocessor**
  - hai_c50_data_prepper, 22
  - hai_cubist_data_prepper, 24
  - hai_data_impute, 26
  - hai_data_poly, 28
  - hai_data_scale, 30
  - hai_data_transform, 32
  - hai_data_trig, 35
hai_earth_data_prepper, 44
hai_glmnet_data_prepper, 54
hai_knn_data_prepper, 70
hai_ranger_data_prepper, 73
hai_svm_poly_data_prepper, 83
hai_svm_rbf_data_prepper, 84
hai_xgboost_data_prepper, 93

* Ranger

hai_auto_ranger, 14
hai_ranger_data_prepper, 73

* Recipes

step_hai_fourier, 96
step_hai_fourier_discrete, 98
step_hai_hyperbolic, 100
step_hai_scale_zero_one, 102
step_hai_scale_zscore, 104
step_hai_winsorized_move, 106
step_hai_winsorized_truncate, 108

* SVM_Poly

hai_auto_svm_poly, 15
hai_svm_poly_data_prepper, 83

* SVM_RBF

hai_auto_svm_rbf, 17
hai_svm_rbf_data_prepper, 84

* Scale

hai_scale_zero_one_augment, 77
hai_scale_zero_one_vec, 78
hai_scale_zscore_augment, 79
hai_scale_zscore_vec, 80
step_hai_scale_zscore, 104

* UMAP

hai_umap_list, 85
hai_umap_plot, 87

* Vector Function

hai_fourier_discrete_vec, 48
hai_fourier_vec, 50
hai_hyperbolic_vec, 58
hai_kurtosis_vec, 71
hai_scale_zero_one_vec, 78
hai_scale_zscore_vec, 80
hai_skewness_vec, 82
hai_winsorized_move_vec, 89
hai_winsorized_truncate_vec, 92

* XBGboost

hai_xgboost_data_prepper, 93

* XGBoost

hai_auto_xgboost, 20

* cubist

hai_auto_cubist, 7
hai_cubist_data_prepper, 24

* glmnet

hai_auto_glmnet, 10

* k-NN

hai_auto_knn, 12

* kNN

hai_glmnet_data_prepper, 54
hai_knn_data_prepper, 70

base::cbind, 61
broom::augment, 67
broom::glance, 67
broom::tidy, 67

color_blind, 3, 76, 77
forcats::as_factor, 61
get_juiced_data, 4, 95
ggrepel::geom_label_repel, 87

h2o::h2o.kmeans, 59
h2o::h2o.predict, 61

hai_auto_c50, 5, 8, 10, 11, 13, 15, 16, 18, 19, 21, 22
hai_auto_cubist, 6, 7, 10, 11, 13, 15, 16, 18, 19, 21, 25
hai_auto_earth, 6, 8, 9, 11, 13, 15, 16, 18, 19, 21, 45
hai_auto_glmnet, 6, 8, 10, 13, 15, 16, 18, 19, 21
hai_auto_knn, 6, 8, 10, 11, 12, 15, 16, 18, 19, 21
hai_auto_ranger, 6, 8, 10, 11, 13, 14, 16, 18, 19, 21, 74
hai_auto_svm_poly, 6, 8, 10, 11, 13, 15, 15, 18, 19, 21, 84
hai_auto_svm_rbf, 6, 8, 10, 11, 13, 15, 16, 17, 19, 21, 85
hai_auto_wflw_metrics, 6, 8, 10, 11, 13, 15, 16, 18, 19, 21
hai_auto_xgboost, 6, 8, 10, 11, 13, 15, 16, 18, 19, 20
hai_c50_data_prepper, 6, 22, 25, 28, 29, 31, 34, 36, 45, 54, 71, 74, 84, 85, 94
hai_control_chart, 23
hai_cubist_data_prepper, 8, 22, 24, 28, 29, 31, 34, 36, 45, 54, 71, 74, 84, 85, 94
hai_data_impute, 22, 25, 26, 29, 31, 34, 36, 45, 54, 71, 74, 84, 85, 94, 95
hai_data_poly, 22, 25, 28, 28, 31, 34, 36, 45, 54, 71, 74, 84, 85, 94, 95
hai_data_scale, 22, 25, 28, 29, 30, 34, 36, 45, 54, 71, 74, 84, 85, 94, 95
hai_data_transform, 22, 25, 28, 29, 31, 32, 36, 45, 54, 71, 74, 84, 85, 94, 95
hai_data_trig, 22, 25, 28, 29, 31, 34, 35, 45, 54, 71, 74, 84, 85, 94, 95
hai_default_classification_metric_set, 37, 38
hai_default_classification_metric_set(), 6, 9, 11, 13, 14, 16, 18, 21
hai_default_regression_metric_set, 37, 38
hai_default_regression_metric_set(), 6, 8, 9, 11, 13, 14, 16, 18, 21
hai_density_hist_plot, 38, 40, 42
hai_density_plot, 39, 39, 42
hai_density_qq_plot, 39, 40, 41
hai_distribution_comparison_tbl, 42, 52, 53
hai_earth_data_prepper, 10, 22, 25, 28, 29, 31, 34, 36, 44, 54, 71, 74, 84, 85, 94
hai_fourier_augment, 45, 48, 57, 73, 77, 80, 89, 91
hai_fourier_augment(), 51
hai_fourier_discrete_augment, 46, 47, 57, 73, 77, 80, 89, 91
hai_fourier_discrete_augment(), 49
hai_fourier_discrete_vec, 48, 51, 58, 72, 78, 81, 83, 90, 93
hai_fourier_vec, 49, 50, 58, 72, 78, 81, 83, 90, 93
hai_get_density_data tbl, 44, 52, 53
hai_get_dist_data_tbl, 44, 52, 53
hai_glmnet_data_prepper, 22, 25, 28, 29, 31, 34, 36, 45, 54, 71, 74, 84, 85, 94
hai_histogram_facet_plot, 55
hai_hyperbolic_augment, 46, 48, 56, 73, 77, 80, 89, 91
hai_hyperbolic_augment(), 58
hai_hyperbolic_vec, 49, 51, 58, 72, 78, 81, 83, 90, 93
hai_kmeans_automl, 59, 62–66, 68, 69
hai_kmeans_automl(), 61
hai_kmeans_automl_predict, 60, 61, 63–66, 68, 69
hai_kmeans_mapped_tbl, 60, 62, 64–66, 68, 69
hai_kmeans_mapped_tbl(), 65, 66, 86
hai_kmeans_obj, 60, 62, 63, 63, 65, 66, 68, 69
hai_kmeans_obj(), 62
hai_kmeans_scree_data_tbl, 60, 62–64, 65, 66, 68, 69
hai_kmeans_scree_plot
(hai_kmeans_scree_plt), 66
hai_kmeans_scree_plt, 60, 62–65, 66, 68, 69
hai_kmeans_scree_plt(), 65
hai_kmeans_tidy_tbl, 60, 62–66, 67, 69
hai_kmeans_user_item_tbl, 60, 62–66, 68, 69
hai_kmeans_user_item_tbl(), 62–64, 67, 86
hai_knn_data_prepper, 22, 25, 28, 29, 31, 34, 36, 45, 54, 70, 74, 84, 85, 94
hai_kurtosis_vec, 49, 51, 58, 71, 78, 81, 83, 90, 93
hai_kurtosis_vec(), 42
hai_polynomial_augment, 46, 48, 57, 72, 77, 80, 89, 91
hai_range_statistic, 75
hai_ranger_data_prepper, 15, 22, 25, 28, 29, 31, 34, 36, 45, 54, 71, 73, 84, 85, 94
hai_scale_color_colorblind, 4, 75, 77
hai_scale_fill_colorblind, 4, 76, 76
hai_scale_zero_one_augment, 46, 48, 57, 73, 77, 78, 80, 81, 89, 91, 105
hai_scale_zero_one_augment(), 78
hai_scale_zero_one_vec, 49, 51, 58, 72, 78, 80, 83, 90, 93, 105
hai_scale_zscore_augment, 46, 48, 57, 73, 77, 79, 81, 89, 91, 105
hai_scale_zscore_augment(), 80
hai_scale_zscore_vec, 49, 51, 58, 72, 77, 78, 80, 83, 90, 93, 105
hai_skewed_features, 81
hai_skewness_vec, 49, 51, 58, 72, 78, 81, 82, 90, 93
hai_skewness_vec(), 42
hai_svm_poly_data_prepper, 16, 22, 25, 28, 29, 31, 34, 36, 45, 54, 71, 74, 83, 85, 94