

# Package ‘alookr’

September 16, 2025

**Type** Package

**Title** Model Classifier for Binary Classification

**Version** 0.4.0

**Date** 2025-09-16

**Description**

A collection of tools that support data splitting, predictive modeling, and model evaluation. A typical function is to split a dataset into a training dataset and a test dataset. Then compare the data distribution of the two datasets. Another feature is to support the development of predictive models and to compare the performance of several predictive models, helping to select the best model.

**Depends** R (>= 3.2.0), ggplot2 (>= 3.0.0), randomForest

**Imports** caTools, cli (>= 1.1.0), dlookr, dplyr (>= 0.7.6), future, MASS, MLmetrics, methods, parallelly, party, purrr, ROCR, ranger, rlang, rpart, stats, tibble, tidyr, tidysselect, xgboost, glmnet

**Suggests** knitr, ISLR, mice, mlbench, rmarkdown, stringi

**Author** Choonghyun Ryu [aut, cre]

**Maintainer** Choonghyun Ryu <choonghyun.ryu@gmail.com>

**BugReports** <https://github.com/choonghyunryu/alookr/issues>

**URL** <https://choonghyunryu.github.io/alookr/>

**License** GPL-2

**Encoding** UTF-8

**VignetteBuilder** knitr

**RoxygenNote** 7.2.3

**Language** en-US

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2025-09-16 02:50:02 UTC

## Contents

cleanse.data.frame . . . . .	2
cleanse.split_df . . . . .	4
compare_diag . . . . .	5
compare_performance . . . . .	7
compare_plot . . . . .	8
compare_target_category . . . . .	9
compare_target_numeric . . . . .	11
extract_set . . . . .	12
matthews . . . . .	13
performance_metric . . . . .	14
plot_cutoff . . . . .	16
plot_performance . . . . .	17
run_models . . . . .	19
run_performance . . . . .	21
run_predict . . . . .	23
sampling_target . . . . .	25
split_by . . . . .	27
summary.split_df . . . . .	28
treatment_corr . . . . .	29
<b>Index</b>	<b>31</b>

---

cleanse.data.frame	<i>Cleansing the dataset for classification modeling</i>
--------------------	--

---

### Description

The `cleanse()` cleanses the dataset for classification modeling

### Usage

```
## S3 method for class 'data.frame'
cleanse(
  .data,
  uniq = TRUE,
  uniq_thres = 0.1,
  char = TRUE,
  missing = FALSE,
  verbose = TRUE,
  ...
)

cleanse(.data, ...)
```

**Arguments**

<code>.data</code>	a data.frame or a <code>tbl_df</code> .
<code>uniq</code>	logical. Set whether to remove the variables whose unique value is one.
<code>uniq_thres</code>	numeric. Set a threshold to removing variables when the ratio of unique values(number of unique values / number of observation) is greater than the set value.
<code>char</code>	logical. Set the change the character to factor.
<code>missing</code>	logical. Set whether to removing variables including missing value
<code>verbose</code>	logical. Set whether to echo information to the console at runtime.
<code>...</code>	further arguments passed to or from other methods.

**Details**

This function is useful when fit the classification model. This function does the following.: Remove the variable with only one value. And remove variables that have a unique number of values relative to the number of observations for a character or categorical variable. In this case, it is a variable that corresponds to an identifier or an identifier. And converts the character to factor.

**Value**

An object of data.frame or train\_df. and return value is an object of the same type as the .data argument.

**Examples**

```
# create sample dataset
set.seed(123L)
id <- sapply(1:1000, function(x)
  paste(c(sample(letters, 5), x), collapse = ""))

year <- "2018"

set.seed(123L)
count <- sample(1:10, size = 1000, replace = TRUE)

set.seed(123L)
alpha <- sample(letters, size = 1000, replace = TRUE)

set.seed(123L)
flag <- sample(c("Y", "N"), size = 1000, prob = c(0.1, 0.9), replace = TRUE)

dat <- data.frame(id, year, count, alpha, flag, stringsAsFactors = FALSE)
# structure of dataset
str(dat)

# cleansing dataset
newDat <- cleanse(dat)

# structure of cleansing dataset
```

```

str(newDat)

# cleansing dataset
newDat <- cleanse(dat, uniq = FALSE)

# structure of cleansing dataset
str(newDat)

# cleansing dataset
newDat <- cleanse(dat, uniq_thres = 0.3)

# structure of cleansing dataset
str(newDat)

# cleansing dataset
newDat <- cleanse(dat, char = FALSE)

# structure of cleansing dataset
str(newDat)

```

---

cleanse.split\_df      *Cleansing the dataset for classification modeling*

---

## Description

Diagnosis of similarity between datasets splitted by train set and set included in the "split\_df" class. and cleansing the "split\_df" class

## Usage

```

## S3 method for class 'split_df'
cleanse(.data, add_character = FALSE, uniq_thres = 0.9, missing = FALSE, ...)

```

## Arguments

.data	an object of class "split_df", usually, a result of a call to split_df().
add_character	logical. Decide whether to include text variables in the compare of categorical data. The default value is FALSE, which also not includes character variables.
uniq_thres	numeric. Set a threshold to removing variables when the ratio of unique values(number of unique values / number of observation) is greater than the set value.
missing	logical. Set whether to removing variables including missing value
...	further arguments passed to or from other methods.

## Details

Remove the detected variables from the diagnosis using the compare\_diag() function.

**Value**

An object of class "split\_df".

**Examples**

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

sb %>%
  cleanse
```

---

compare\_diag

*Diagnosis of train set and test set of split\_df object*

---

**Description**

Diagnosis of similarity between datasets splitted by train set and set included in the "split\_df" class.

**Usage**

```
compare_diag(
  .data,
  add_character = FALSE,
  uniq_thres = 0.01,
  miss_msg = TRUE,
  verbose = TRUE
)
```

**Arguments**

.data	an object of class "split_df", usually, a result of a call to split_df().
add_character	logical. Decide whether to include text variables in the compare of categorical data. The default value is FALSE, which also not includes character variables.
uniq_thres	numeric. Set a threshold to removing variables when the ratio of unique values(number of unique values / number of observation) is greater than the set value.
miss_msg	logical. Set whether to output a message when diagnosing missing value.
verbose	logical. Set whether to echo information to the console at runtime.

## Details

In the two split datasets, a variable with a single value, a variable with a level not found in any dataset, and a variable with a high ratio to the number of levels are diagnosed.

## Value

list. Variables of tbl\_df for first component named "single\_value":

- variables : character. variable name
- train\_uniq : character. the type of unique value in train set. it is divided into "single" and "multi".
- test\_uniq : character. the type of unique value in test set. it is divided into "single" and "multi".

Variables of tbl\_df for second component named "uniq\_rate":

- variables : character. categorical variable name
- train\_uniqcount : numeric. the number of unique value in train set
- train\_uniqrate : numeric. the ratio of unique values(number of unique values / number of observation) in train set
- test\_uniqcount : numeric. the number of unique value in test set
- test\_uniqrate : numeric. the ratio of unique values(number of unique values / number of observation) in test set

Variables of tbl\_df for third component named "missing\_level":

- variables : character. variable name
- n\_levels : integer. count of level of categorical variable
- train\_missing\_nlevel : integer. the number of non-existent levels in the train set
- test\_missing\_nlevel : integer. the number of non-existent levels in the test set

## Examples

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

defaults <- ISLR::Default
defaults$id <- seq(NROW(defaults))

set.seed(1)
defaults[sample(seq(NROW(defaults)), 3), "student"] <- NA
set.seed(2)
defaults[sample(seq(NROW(defaults)), 10), "balance"] <- NA

sb <- defaults %>%
  split_by(default)

sb %>%
```

```
compare_diag()

sb %>%
  compare_diag(add_character = TRUE)

sb %>%
  compare_diag(uniq_thres = 0.0005)
```

---

compare\_performance    *Compare model performance*

---

## Description

compare\_performance() compares the performance of a model with several model performance metrics.

## Usage

```
compare_performance(model)
```

## Arguments

model                    A model\_df. results of predicted model that created by run\_predict().

## Value

list. results of compared model performance. list has the following components:

- recommend\_model : character. The name of the model that is recommended as the best among the various models.
- top\_count : numeric. The number of best performing performance metrics by model.
- mean\_rank : numeric. Average of ranking individual performance metrics by model.
- top\_metric : list. The name of the performance metric with the best performance on individual performance metrics by model.

The performance metrics calculated are as follows.:

- ZeroOneLoss : Normalized Zero-One Loss(Classification Error Loss).
- Accuracy : Accuracy.
- Precision : Precision.
- Recall : Recall.
- Specificity : Specificity.
- F1\_Score : F1 Score.
- LogLoss : Log loss / Cross-Entropy Loss.
- AUC : Area Under the Receiver Operating Characteristic Curve (ROC AUC).

- Gini : Gini Coefficient.
- PRAUC : Area Under the Precision-Recall Curve (PR AUC).
- LiftAUC : Area Under the Lift Chart.
- GainAUC : Area Under the Gain Chart.
- KS\_Stat : Kolmogorov-Smirnov Statistic.

### Examples

```
library(dplyr)

# Divide the train data set and the test data set.
sb <- rpart::kyphosis %>%
  split_by(Kyphosis)

# Extract the train data set from original data set.
train <- sb %>%
  extract_set(set = "train")

# Extract the test data set from original data set.
test <- sb %>%
  extract_set(set = "test")

# Sampling for unbalanced data set using SMOTE(synthetic minority over-sampling technique).
train <- sb %>%
  sampling_target(seed = 1234L, method = "ubSMOTE")

# Cleaning the set.
train <- train %>%
  cleanse

# Run the model fitting.
result <- run_models(.data = train, target = "Kyphosis", positive = "present")

# Predict the model.
pred <- run_predict(result, test)

# Compare the model performance
compare_performance(pred)
```

---

compare\_plot

*Comparison plot of train set and test set*

---

### Description

Plot compare information of the train set and test set included in the "split\_df" class.



**Usage**

```
compare_plot(.data, ...)
```

**Arguments**

`.data` an object of class "split\_df", usually, a result of a call to `split_df()`.

`...` one or more unquoted expressions separated by commas. Select the variable you want to plotting. You can treat variable names like they are positions. Positive values select variables; negative values to drop variables. If the first expression is negative, `compare_target_category()` will automatically start with all variables. These arguments are automatically quoted and evaluated in a context where column names represent column positions. They support unquoting and splicing.

**Details**

The numerical variables are density plots and the categorical variables are mosaic plots to compare the distribution of train sets and test sets.

**Value**

There is no return value. Draw only the plot.

**Examples**

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

sb %>%
  compare_plot("income")

sb %>%
  compare_plot()
```

---

compare\_target\_category

*Comparison of categorical variables of train set and test set*

---

**Description**

Compare the statistics of the categorical variables of the train set and test set included in the "split\_df" class.

**Usage**

```
compare_target_category(.data, ..., add_character = FALSE, margin = FALSE)
```

**Arguments**

<code>.data</code>	an object of class "split_df", usually, a result of a call to <code>split_df()</code> .
<code>...</code>	one or more unquoted expressions separated by commas. Select the categorical variable you want to compare. You can treat variable names like they are positions. Positive values select variables; negative values to drop variables. If the first expression is negative, <code>compare_target_category()</code> will automatically start with all variables. These arguments are automatically quoted and evaluated in a context where column names represent column positions. They support unquoting and splicing.
<code>add_character</code>	logical. Decide whether to include text variables in the compare of categorical data. The default value is <code>FALSE</code> , which also not includes character variables.
<code>margin</code>	logical. Choose to calculate the marginal frequency information.

**Details**

Compare the statistics of the numerical variables of the train set and the test set to determine whether the raw data is well separated into two data sets.

**Value**

`tbl_df`. Variables of `tbl_df` for comparison:

- `variable` : character. categorical variable name
- `level` : factor. level of categorical variables
- `train` : numeric. the relative frequency of the level in the train set
- `test` : numeric. the relative frequency of the level in the test set
- `abs_diff` : numeric. the absolute value of the difference between two relative frequencies

**Examples**

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

sb %>%
  compare_target_category()

sb %>%
  compare_target_category(add_character = TRUE)
```

```

sb %>%
  compare_target_category(margin = TRUE)

sb %>%
  compare_target_category(student)

sb %>%
  compare_target_category(student, margin = TRUE)

```

---

compare\_target\_numeric

*Comparison of numerical variables of train set and test set*

---

### Description

Compare the statistics of the numerical variables of the train set and test set included in the "split\_df" class.

### Usage

```
compare_target_numeric(.data, ...)
```

### Arguments

.data	an object of class "split_df", usually, a result of a call to split_df().
...	one or more unquoted expressions separated by commas. Select the numeric variable you want to compare. You can treat variable names like they are positions. Positive values select variables; negative values to drop variables. If the first expression is negative, compare_target_numeric() will automatically start with all variables. These arguments are automatically quoted and evaluated in a context where column names represent column positions. They support unquoting and splicing.

### Details

Compare the statistics of the numerical variables of the train set and the test set to determine whether the raw data is well separated into two data sets.

### Value

tbl\_df. Variables for comparison:

- variable : character. numeric variable name
- train\_mean : numeric. arithmetic mean of train set
- test\_mean : numeric. arithmetic mean of test set

- train\_sd : numeric. standard deviation of train set
- test\_sd : numeric. standard deviation of test set
- train\_z : numeric. the arithmetic mean of the train set divided by the standard deviation
- test\_z : numeric. the arithmetic mean of the test set divided by the standard deviation

### Examples

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

sb %>%
  compare_target_numeric()

sb %>%
  compare_target_numeric(balance)
```

---

extract\_set

*Extract train/test dataset*

---

### Description

Extract train set or test set from split\_df class object

### Usage

```
extract_set(x, set = c("train", "test"))
```

### Arguments

x	an object of class "split_df", usually, a result of a call to split_df().
set	character. Specifies whether the extracted data is a train set or a test set. You can use "train" or "test".

### Details

Extract the train or test sets based on the parameters you defined when creating split\_df with split\_by().

### Value

an object of class "tbl\_df".

**Examples**

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

train <- sb %>%
  extract_set(set = "train")

test <- sb %>%
  extract_set(set = "test")
```

---

matthews

*Compute Matthews Correlation Coefficient*

---

**Description**

compute the Matthews correlation coefficient with actual and predict values.

**Usage**

```
matthews(predicted, y, positive)
```

**Arguments**

predicted	numeric. the predicted value of binary classification
y	factor or character. the actual value of binary classification
positive	level of positive class of binary classification

**Details**

The Matthews Correlation Coefficient has a value between -1 and 1, and the closer to 1, the better the performance of the binary classification.

**Value**

numeric. The Matthews Correlation Coefficient.

**Examples**

```

# simulate actual data
set.seed(123L)
actual <- sample(c("Y", "N"), size = 100, prob = c(0.3, 0.7), replace = TRUE)
actual

# simulate predict data
set.seed(123L)
pred <- sample(c("Y", "N"), size = 100, prob = c(0.2, 0.8), replace = TRUE)
pred

# simulate confusion matrix
table(pred, actual)

matthews(pred, actual, "Y")

```

---

performance\_metric      *Calculate metrics for model evaluation*

---

**Description**

Calculate some representative metrics for binary classification model evaluation.

**Usage**

```

performance_metric(
  pred,
  actual,
  positive,
  metric = c("ZeroOneLoss", "Accuracy", "Precision", "Recall", "Sensitivity",
    "Specificity", "F1_Score", "Fbeta_Score", "LogLoss", "AUC", "Gini", "PRAUC",
    "LiftAUC", "GainAUC", "KS_Stat", "ConfusionMatrix"),
  cutoff = 0.5,
  beta = 1
)

```

**Arguments**

pred	numeric. Probability values that predicts the positive class of the target variable.
actual	factor. The value of the actual target variable.
positive	character. Level of positive class of binary classification.
metric	character. The performance metrics you want to calculate. See details.
cutoff	numeric. Threshold for classifying predicted probability values into positive and negative classes.
beta	numeric. Weight of precision in harmonic mean for F-Beta Score.

## Details

The cutoff argument applies only if the metric argument is "ZeroOneLoss", "Accuracy", "Precision", "Recall", "Sensitivity", "Specificity", "F1\_Score", "Fbeta\_Score", "ConfusionMatrix".

## Value

numeric or table object. Confusion Matrix return by table object. and otherwise is numeric.: The performance metrics calculated are as follows.:

- ZeroOneLoss : Normalized Zero-One Loss(Classification Error Loss).
- Accuracy : Accuracy.
- Precision : Precision.
- Recall : Recall.
- Sensitivity : Sensitivity.
- Specificity : Specificity.
- F1\_Score : F1 Score.
- Fbeta\_Score : F-Beta Score.
- LogLoss : Log loss / Cross-Entropy Loss.
- AUC : Area Under the Receiver Operating Characteristic Curve (ROC AUC).
- Gini : Gini Coefficient.
- PRAUC : Area Under the Precision-Recall Curve (PR AUC).
- LiftAUC : Area Under the Lift Chart.
- GainAUC : Area Under the Gain Chart.
- KS\_Stat : Kolmogorov-Smirnov Statistic.
- ConfusionMatrix : Confusion Matrix.

## Examples

```
library(dplyr)

# Divide the train data set and the test data set.
sb <- rpart::kyphosis %>%
  split_by(Kyphosis)

# Extract the train data set from original data set.
train <- sb %>%
  extract_set(set = "train")

# Extract the test data set from original data set.
test <- sb %>%
  extract_set(set = "test")

# Sampling for unbalanced data set using SMOTE(synthetic minority over-sampling technique).
train <- sb %>%
  sampling_target(seed = 1234L, method = "ubSMOTE")
```

```

# Cleaning the set.
train <- train %>%
  cleanse

# Run the model fitting.
result <- run_models(.data = train, target = "Kyphosis", positive = "present")
result

# Predict the model.
pred <- run_predict(result, test)
pred

# Calculate Accuracy.
performance_metric(attr(pred$predicted[[1]], "pred_prob"), test$Kyphosis,
  "present", "Accuracy")
# Calculate Confusion Matrix.
performance_metric(attr(pred$predicted[[1]], "pred_prob"), test$Kyphosis,
  "present", "ConfusionMatrix")
# Calculate Confusion Matrix by cutoff = 0.55.
performance_metric(attr(pred$predicted[[1]], "pred_prob"), test$Kyphosis,
  "present", "ConfusionMatrix", cutoff = 0.55)

```

---

plot\_cutoff

*Visualization for cut-off selection*


---

## Description

plot\_cutoff() visualizes a plot to select a cut-off that separates positive and negative from the probabilities that are predictions of a binary classification, and suggests a cut-off.

## Usage

```

plot_cutoff(
  predicted,
  y,
  positive,
  type = c("mcc", "density", "prob"),
  measure = c("mcc", "cross", "half")
)

```

## Arguments

predicted	numeric. the predicted value of binary classification
y	factor or character. the actual value of binary classification
positive	level of positive class of binary classification



type	character. Visualization type. "mcc" draw the Matthews Correlation Coefficient scatter plot, "density" draw the density plot of negative and positive, and "prob" draws line or points plots of the predicted probability.
measure	character. The kind of measure that calculates the cutoff. "mcc" is the Matthews Correlation Coefficient, "cross" is the point where the positive and negative densities cross, and "half" is the median of the probability, 0.5

### Details

If the type argument is "prob", visualize the points plot if the number of observations is less than 100. If the observation is greater than 100, draw a line plot. In this case, the speed of visualization can be slow.

### Value

numeric. cut-off value

### Examples

```
library(ggplot2)
library(rpart)
data(kyphosis)

fit <- glm(Kyphosis ~., family = binomial, kyphosis)
pred <- predict(fit, type = "response")

cutoff <- plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "mcc")
cutoff
plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "mcc", measure = "cross")
plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "mcc", measure = "half")

plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "density", measure = "mcc")
plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "density", measure = "cross")
plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "density", measure = "half")

plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "prob", measure = "mcc")
plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "prob", measure = "cross")
plot_cutoff(pred, kyphosis$Kyphosis, "present", type = "prob", measure = "half")
```

---

plot\_performance

*Visualization for ROC curve*

---

### Description

plot\_performance() visualizes a plot to ROC curve that separates model algorithm.

**Usage**

```
plot_performance(model)
```

**Arguments**

model                    A model\_df. results of predicted model that created by run\_predict().

**Details**

The ROC curve is output for each model included in the model\_df class object specified as a model argument.

**Value**

There is no return value. Only the plot is drawn.

**Examples**

```
library(dplyr)

# Divide the train data set and the test data set.
sb <- rpart::kyphosis %>%
  split_by(Kyphosis)

# Extract the train data set from original data set.
train <- sb %>%
  extract_set(set = "train")

# Extract the test data set from original data set.
test <- sb %>%
  extract_set(set = "test")

# Sampling for unbalanced data set using SMOTE(synthetic minority over-sampling technique).
train <- sb %>%
  sampling_target(seed = 1234L, method = "ubSMOTE")

# Cleaning the set.
train <- train %>%
  cleanse

# Run the model fitting.
result <- run_models(.data = train, target = "Kyphosis", positive = "present")

# Predict the model.
pred <- run_predict(result, test)

# Plot ROC curve
plot_performance(pred)
```

---

`run_models`*Fit binary classification model*

---

## Description

Fit some representative binary classification models.

## Usage

```
run_models(  
  .data,  
  target,  
  positive,  
  models = c("logistic", "rpart", "ctree", "randomForest", "ranger", "xgboost",  
             "lasso")  
)
```

## Arguments

<code>.data</code>	A <code>train_df</code> . Train data to fit the model. It also supports <code>tbl_df</code> , <code>tbl</code> , and <code>data.frame</code> objects.
<code>target</code>	character. Name of target variable.
<code>positive</code>	character. Level of positive class of binary classification.
<code>models</code>	character. Algorithm types of model to fit. See details. default value is <code>c("logistic", "rpart", "ctree", "randomForest", "ranger", "lasso")</code> .

## Details

Supported models are functions supported by the representative model package used in R environment. The following binary classifications are supported:

- "logistic" : logistic regression by `glm()` in stats package.
- "rpart" : recursive partitioning tree model by `rpart()` in rpart package.
- "ctree" : conditional inference tree model by `ctree()` in party package.
- "randomForest" : random forest model by `randomForest()` in randomForest package.
- "ranger" : random forest model by `ranger()` in ranger package.
- "xgboost" : XGBoosting model by `xgboost()` in xgboost package.
- "lasso" : lasso model by `glmnet()` in glmnet package.

`run_models()` executes the process in parallel when fitting the model. However, it is not supported in MS-Windows operating system and RStudio environment.

**Value**

model\_df. results of fitted model. model\_df is composed of tbl\_df and contains the following variables.:

- step : character. The current stage in the model fit process. The result of calling run\_models() is returned as "1.Fitted".
- model\_id : character. Type of fit model.
- target : character. Name of target variable.
- is\_factor : logical. Indicates whether the target variable is a factor.
- positive : character. Level of positive class of binary classification.
- negative : character. Level of negative class of binary classification.
- fitted\_model : list. Fitted model object.

**Examples**

```
library(dplyr)

# Divide the train data set and the test data set.
sb <- rpart::kyphosis %>%
  split_by(Kyphosis)

# Extract the train data set from original data set.
train <- sb %>%
  extract_set(set = "train")

# Extract the test data set from original data set.
test <- sb %>%
  extract_set(set = "test")

# Sampling for unbalanced data set using SMOTE(synthetic minority over-sampling technique).
train <- sb %>%
  sampling_target(seed = 1234L, method = "ubSMOTE")

# Cleaning the set.
train <- train %>%
  cleanse

# Run the model fitting.
result <- run_models(.data = train, target = "Kyphosis", positive = "present")
result

# Run the several kinds model fitting by dplyr
train %>%
  run_models(target = "Kyphosis", positive = "present")

# Run the logistic model fitting by dplyr
train %>%
  run_models(target = "Kyphosis", positive = "present", models = "logistic")
```

---

run_performance	<i>Apply calculate performance metrics for model evaluation</i>
-----------------	---

---

**Description**

Apply calculate performance metrics for binary classification model evaluation.

**Usage**

```
run_performance(model, actual = NULL)
```

**Arguments**

model	A model_df. results of predicted model that created by run_predict().
actual	factor. A data of target variable to evaluate the model. It supports factor that has binary class.

**Details**

run\_performance() is performed in parallel when calculating the performance evaluation index. However, it is not supported in MS-Windows operating system and RStudio environment.

**Value**

model\_df. results of predicted model. model\_df is composed of tbl\_df and contains the following variables.:

- step : character. The current stage in the model fit process. The result of calling run\_performance() is returned as "3.Performanced".
- model\_id : character. Type of fit model.
- target : character. Name of target variable.
- positive : character. Level of positive class of binary classification.
- fitted\_model : list. Fitted model object.
- predicted : list. Predicted value by individual model. Each value has a predict\_class class object.
- performance : list. Calculate metrics by individual model. Each value has a numeric vector.

The performance metrics calculated are as follows.:

- ZeroOneLoss : Normalized Zero-One Loss(Classification Error Loss).
- Accuracy : Accuracy.
- Precision : Precision.
- Recall : Recall.
- Sensitivity : Sensitivity.
- Specificity : Specificity.

- F1\_Score : F1 Score.
- Fbeta\_Score : F-Beta Score.
- LogLoss : Log loss / Cross-Entropy Loss.
- AUC : Area Under the Receiver Operating Characteristic Curve (ROC AUC).
- Gini : Gini Coefficient.
- PRAUC : Area Under the Precision-Recall Curve (PR AUC).
- LiftAUC : Area Under the Lift Chart.
- GainAUC : Area Under the Gain Chart.
- KS\_Stat : Kolmogorov-Smirnov Statistic.

### Examples

```

library(dplyr)

# Divide the train data set and the test data set.
sb <- rpart::kyphosis %>%
  split_by(Kyphosis)

# Extract the train data set from original data set.
train <- sb %>%
  extract_set(set = "train")

# Extract the test data set from original data set.
test <- sb %>%
  extract_set(set = "test")

# Sampling for unbalanced data set using SMOTE(synthetic minority over-sampling technique).
train <- sb %>%
  sampling_target(seed = 1234L, method = "ubSMOTE")

# Cleaning the set.
train <- train %>%
  cleanse

# Run the model fitting.
result <- run_models(.data = train, target = "Kyphosis", positive = "present")
result

# Predict the model. (Case 1)
pred <- run_predict(result, test)
pred

# Calculate performance metrics. (Case 1)
perf <- run_performance(pred)
perf
perf$performance

# Predict the model. (Case 2)
pred <- run_predict(result, test[, -1])

```

```

pred

# Calculate performance metrics. (Case 2)
perf <- run_performance(pred, pull(test[, 1]))
perf
perf$performance

# Convert to matrix for compare performance.
sapply(perf$performance, "c")

```

---

run\_predict

*Predict binary classification model*


---

## Description

Predict some representative binary classification models.

## Usage

```
run_predict(model, .data, cutoff = 0.5)
```

## Arguments

model	A model_df. results of fitted model that created by run_models().
.data	A tbl_df. The data set to predict the model. It also supports tbl, and data.frame objects.
cutoff	numeric. Cut-off that determines the positive from the probability of predicting the positive.

## Details

Supported models are functions supported by the representative model package used in R environment. The following binary classifications are supported:

- "logistic" : logistic regression by predict.glm() in stats package.
- "rpart" : recursive partitioning tree model by predict.rpart() in rpart package.
- "ctree" : conditional inference tree model by predict() in stats package.
- "randomForest" : random forest model by predict.randomForest() in randomForest package.
- "ranger" : random forest model by predict.ranger() in ranger package.
- "xgboost" : random forest model by predict.xgb.Booster() in xgboost package.
- "lasso" : random forest model by predict.glmnet() in glmnet package.

run\_predict() is executed in parallel when predicting by model. However, it is not supported in MS-Windows operating system and RStudio environment.

**Value**

model\_df. results of predicted model. model\_df is composed of tbl\_df and contains the following variables.:

- step : character. The current stage in the model fit process. The result of calling run\_predict() is returned as "2.Predicted".
- model\_id : character. Type of fit model.
- target : character. Name of target variable.
- is\_factor : logical. Indicates whether the target variable is a factor.
- positive : character. Level of positive class of binary classification.
- negative : character. Level of negative class of binary classification.
- fitted\_model : list. Fitted model object.
- predicted : list. Predicted value by individual model. Each value has a predict\_class class object.

**Examples**

```
library(dplyr)

# Divide the train data set and the test data set.
sb <- rpart::kyphosis %>%
  split_by(Kyphosis)

# Extract the train data set from original data set.
train <- sb %>%
  extract_set(set = "train")

# Extract the test data set from original data set.
test <- sb %>%
  extract_set(set = "test")

# Sampling for unbalanced data set using SMOTE(synthetic minority over-sampling technique).
train <- sb %>%
  sampling_target(seed = 1234L, method = "ubSMOTE")

# Cleaning the set.
train <- train %>%
  cleanse

# Run the model fitting.
result <- run_models(.data = train, target = "Kyphosis", positive = "present")
result

# Predict the model.
pred <- run_predict(result, test)
pred

# Run the several kinds model predict by dplyr
result %>%
```



```
run_predict(test)
```

---

sampling_target	<i>Extract the data to fit the model</i>
-----------------	--

---

### Description

To solve the imbalanced class, perform sampling in the train set of split\_df.

### Usage

```
sampling_target(
  .data,
  method = c("ubUnder", "ubOver", "ubSMOTE"),
  seed = NULL,
  perc = 50,
  k = ifelse(method == "ubSMOTE", 5, 0),
  perc.over = 200,
  perc.under = 200
)
```

### Arguments

.data	an object of class "split_df", usually, a result of a call to split_df().
method	character. sampling methods. "ubUnder" is under-sampling, and "ubOver" is over-sampling, "ubSMOTE" is SMOTE(Synthetic Minority Over-sampling TEchnique).
seed	integer. random seed used for sampling
perc	integer. The percentage of positive class in the final dataset. It is used only in under-sampling. The default is 50. perc can not exceed 50.
k	integer. It is used only in over-sampling and SMOTE. If over-sampling and if K=0: sample with replacement from the minority class until we have the same number of instances in each class. under-sampling and if K>0: sample with replacement from the minority class until we have k-times the original number of minority instances. If SMOTE, the number of neighbours to consider as the pool from where the new examples are generated
perc.over	integer. It is used only in SMOTE. perc.over/100 is the number of new instances generated for each rare instance. If perc.over < 100 a single instance is generated.
perc.under	integer. It is used only in SMOTE. perc.under/100 is the number of "normal" (majority class) instances that are randomly selected for each smoted observation.

## Details

In order to solve the problem of imbalanced class, sampling is performed by under sampling, over sampling, SMOTE method.

## Value

An object of train\_df.

## attributes of train\_df class

The attributes of the train\_df class are as follows.:

- sample\_seed : integer. random seed used for sampling
- method : character. sampling methods.
- perc : integer. perc argument value
- k : integer. k argument value
- perc.over : integer. perc.over argument value
- perc.under : integer. perc.under argument value
- binary : logical. whether the target variable is a binary class
- target : character. target variable name
- minority : character. the level of the minority class
- majority : character. the level of the majority class

## Examples

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

# under-sampling with random seed
under <- sb %>%
  sampling_target(seed = 1234L)

under %>%
  count(default)

# under-sampling with random seed, and minority class frequency is 40%
under40 <- sb %>%
  sampling_target(seed = 1234L, perc = 40)

under40 %>%
  count(default)
```

```

# over-sampling with random seed
over <- sb %>%
  sampling_target(method = "ubOver", seed = 1234L)

over %>%
  count(default)

# over-sampling with random seed, and k = 10
over10 <- sb %>%
  sampling_target(method = "ubOver", seed = 1234L, k = 10)

over10 %>%
  count(default)

# SMOTE with random seed
smote <- sb %>%
  sampling_target(method = "ubSMOTE", seed = 1234L)

smote %>%
  count(default)

# SMOTE with random seed, and perc.under = 250
smote250 <- sb %>%
  sampling_target(method = "ubSMOTE", seed = 1234L, perc.under = 250)

smote250 %>%
  count(default)

```

---

split\_by

*Split Data into Train and Test Set*


---

### Description

The `split_by()` splits the `data.frame` or `tbl_df` into a train set and a test set.

### Usage

```
split_by(.data, ...)
```

```
## S3 method for class 'data.frame'
split_by(.data, target, ratio = 0.7, seed = NULL, ...)
```

### Arguments

<code>.data</code>	a <code>data.frame</code> or a <code>tbl_df</code> .
<code>...</code>	further arguments passed to or from other methods.
<code>target</code>	unquoted expression or variable name. the name of the target variable
<code>ratio</code>	numeric. the ratio of the train dataset. default is 0.7
<code>seed</code>	random seed used for splitting

**Details**

The `split_df` class is created, which contains the split information and criteria to separate the training and the test set.

**Value**

An object of `split_by`.

**attributes of `split_by`**

The attributes of the `split_df` class are as follows.:

- `split_seed` : integer. random seed used for splitting
- `target` : character. the name of the target variable
- `binary` : logical. whether the target variable is binary class
- `minority` : character. the name of the minority class
- `majority` : character. the name of the majority class
- `minority_rate` : numeric. the rate of the minority class
- `majority_rate` : numeric. the rate of the majority class

**Examples**

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

sb
```

---

summary.split\_df      *Summarizing split\_df information*

---

**Description**

summary method for "split\_df" class.

**Usage**

```
## S3 method for class 'split_df'
summary(object, ...)
```

**Arguments**

object            an object of class "split\_df", usually, a result of a call to split\_df().  
 ...               further arguments passed to or from other methods.

**Details**

summary.split\_df provides information on the number of two split data sets, minority class and majority class.

**Value**

NULL is returned. However, the split train set and test set information are displayed. The output information is as follows.:

- Random seed
- Number of train sets and test sets
- Name of target variable
- Target variable minority class and majority class information (label and ratio)

**Examples**

```
library(dplyr)

# Credit Card Default Data
head(ISLR::Default)

# Generate data for the example
sb <- ISLR::Default %>%
  split_by(default)

sb
summary(sb)
```

---

treatment\_corr

*Diagnosis and removal of highly correlated variables*


---

**Description**

The treatment\_corr() diagnose pairs of highly correlated variables or remove on of them.

**Usage**

```
treatment_corr(.data, corr_thres = 0.8, treat = TRUE, verbose = TRUE)
```

**Arguments**

<code>.data</code>	a data.frame or a <code>tbl_df</code> .
<code>corr_thres</code>	numeric. Set a threshold to detecting variables when correlation greater then threshold.
<code>treat</code>	logical. Set whether to removing variables
<code>verbose</code>	logical. Set whether to echo information to the console at runtime.

**Details**

The correlation coefficient of pearson is obtained for continuous variables and the correlation coefficient of spearman for categorical variables.

**Value**

An object of data.frame or train\_df. and return value is an object of the same type as the .data argument. However, several variables can be excluded by correlation between variables.

**Examples**

```
# numerical variable
x1 <- 1:100
set.seed(12L)
x2 <- sample(1:3, size = 100, replace = TRUE) * x1 + rnorm(1)
set.seed(1234L)
x3 <- sample(1:2, size = 100, replace = TRUE) * x1 + rnorm(1)

# categorical variable
x4 <- factor(rep(letters[1:20], time = 5))
set.seed(100L)
x5 <- factor(rep(letters[1:20 + sample(1:6, size = 20, replace = TRUE)], time = 5))
set.seed(200L)
x6 <- factor(rep(letters[1:20 + sample(1:3, size = 20, replace = TRUE)], time = 5))
set.seed(300L)
x7 <- factor(sample(letters[1:5], size = 100, replace = TRUE))

exam <- data.frame(x1, x2, x3, x4, x5, x6, x7)
str(exam)
head(exam)

# default case
treatment_corr(exam)

# not removing variables
treatment_corr(exam, treat = FALSE)

# Set a threshold to detecting variables when correlation greater then 0.9
treatment_corr(exam, corr_thres = 0.9, treat = FALSE)

# not verbose mode
treatment_corr(exam, verbose = FALSE)
```

# Index

`cleanse (cleanse.data.frame)`, 2  
`cleanse.data.frame`, 2  
`cleanse.split_df`, 4  
`compare_diag`, 5  
`compare_performance`, 7  
`compare_plot`, 8  
`compare_target_category`, 9  
`compare_target_numeric`, 11

`extract_set`, 12

`matthews`, 13

`performance_metric`, 14  
`plot_cutoff`, 16  
`plot_performance`, 17

`run_models`, 19  
`run_performance`, 21  
`run_predict`, 23

`sampling_target`, 25  
`split_by`, 27  
`summary.split_df`, 28

`tbl_df`, 3, 27, 30  
`treatment_corr`, 29