Package ‘helda’

January 31, 2020

Title Preprocess Data and Get Better Insights from Machine Learning Models

Description The main focus is on preprocessing and data visualization of machine learning models performances.
Some functions allow to fill gaps in time series using linear interpolation on panel data, some functions permit to draw lift effect and lift curves in order to benchmark machine learning models or you can even find the optimal number of clusters in agglomerative clustering algorithm.

Depends R (>= 3.5.0),
Imports dplyr(>= 0.7.8), ggplot2(>= 3.1.0), sqldf(>= 0.4-11), stringr(>= 1.3.1), rlang(>= 0.4.2), stats(>= 3.5.0)
Suggests devtools (>= 2.2.1), testthat (>= 2.1.0), covr (>= 3.4.0)
License GPL-3
Encoding UTF-8
LazyData true

URL https://www.github.com/Redcart/helda
BugReports https://github.com/Redcart/helda/issues
RoxygenNote 7.0.1
NeedsCompilation no
Author Simon Corde [aut, cre]
Maintainer Simon Corde <simon.corde@hotmail.fr>
Repository CRAN
Date/Publication 2020-01-31 15:20:02 UTC

R topics documented:

clust_centroid ............................................................. 2
compute_inertia .......................................................... 3
compute_inertia_ahc ..................................................... 3
clust_centroid

Centroid of a cluster

Description

This function allows to compute the centroid of a cluster in a R data frame

Usage

clust_centroid(i, data, cluster)

Arguments

i integer that represents the cluster number
data R data frame (all columns are required to be numeric types)
cluster character representing the column name of the data frame representing the clusters

Value

a vector of coordinates for the centroid of the cluster i

Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://www.github.com/Redcart/helda
compute_inertia

Inertia of a data frame

Description
This function allows to compute the inertia of a R data frame

Usage
compute_inertia(data)

Arguments
data R data frame (all columns are required to be numeric types)

Value
a numeric value representing the total inertia

Author(s)
Simon CORDE

References
Link to the author’s github package repository: https://www.github.com/Redcart/helda

Examples
compute_inertia(mtcars)

compute_inertia_ahc

Intragroup inertia for choosing the optimal number of clusters in Agglomerative Clustering

Description
This function allows to compute the inter group inertia from agglomerative clustering for different number of clusters

Usage
compute_inertia_ahc(data, method = "ward.D", max_clusters = 10)
create_calendar

Arguments

- **data**: R data frame (all columns are required to be numeric types)
- **method**: character that specifies the method on which the agglomerative is built upon (by default set to "ward.D")
- **max_clusters**: integer that represents the maximal number of clusters for which we intend to compute intra group inertia

Value

- a vector of length `max_clusters` containing the inter group inertia for agglomerative clustering. The ith value of the vector corresponds to the inter group inertia from agglomerative clustering run with i clusters.

References

Link to the author's github package repository: [https://www.github.com/Redcart/helda](https://www.github.com/Redcart/helda)

Examples

```r
data = iris[,1:4]
result <- compute_inertia_ahc(data=data, max_clusters=15)
```

create_calendar

**Complete empty calendar**

Description

This function allows to create a complete empty calendar on a year scale

Usage

```r
create_calendar(data, key_variable, time_variable, start_year, end_year)
```

Arguments

- **data**: R data frame
- **key_variable**: character that represents the variable name that refers to the key variable in the panel data (ID, ...)
- **time_variable**: character that represents the variable name that permits to sort observation on a time scale
- **start_year**: integer of the starting year of the time serie
- **end_year**: integer of the ending year of the time serie

Value

- a R data frame with the key and time variable. Each id key is associated with all years between `start_year` and `end_year`
create_formula

Author(s)
Simon CORDE

References
Link to the author's github package repository: https://www.github.com/Redcart/helda

Examples
rep(c("Paris", "Madrid", "Berlin"), each = 10)

jeu_donnees <- data.frame("country" = rep(c("France", "Spain", "Germany"), each = 10),
"capital" = rep(c("Paris", "Madrid", "Berlin"), each = 10),
"year" = 2009:2018,
"gdp" = c(NA, NA, 200, 300, 500, 1000, NA, NA, NA, 500,
0, NA, NA, NA, NA, NA, NA, 800, 1200, 1500,
100, 200, 400, 700, 700, 800, 600, 500, NA, NA))

jeu_donnees <- na.omit(jeu_donnees)# we artificially create some gaps in the time series
data_1 <- create_calendar(data = jeu_donnees, key_variable = "country", time_variable = "year",
start_year = 2009, end_year = 2018)
data_2 <- start_end_to_fill(data = jeu_donnees, calendar = data_1, gap_variable = "gdp",
key_variable = "country", time_variable = "year")
data_3 <- gap_to_fill(data = data_2, gap_variable = "gdp_corrected_1", key_variable = "country",
time_variable = "year", digits = 1)

create_formula

Create a formula

Description
This function allows to create a formula from the columns of a data frame very quickly

Usage
create_formula(data, position = 1)

Arguments
data R data frame
position integer representing the number of the column in the data frame that we want to create a formula

Value
string that contains the formula The formula is displayed with the following format: "Y ~ X1 + X2 + ...

Author(s)
Simon CORDE
References

Link to the author’s github package repository: https://www.github.com/Redcart/helda

Examples

data <- iris
str(data)
result <- create_formula(data = data, position = 4)

---

gap_to_fill Filling intermediate gaps in a time serie

Description

This function allows to fill intermediate gaps in panel data by linear interpolation

Usage

gap_to_fill(data, gap_variable, key_variable, time_variable, digits = 2)

Arguments

data R data frame

gap_variable character name of the variable we want to fill the start and end gaps
key_variable character variable name that refers to the key variable in the panel data (ID, ...)
time_variable character time variable name that permits to sort observation on a time scale
digits integer number of decimals to keep for the rounding (by default set to 2)

Value

a R data frame containing the original columns and a new one:

• gap_variable_corrected_2: the gap variable with intermediate gaps filled

Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://www.github.com/Redcart/helda
Examples

```r
rep(c("Paris", "Madrid", "Berlin"), each = 10)
jeu_donnees <- data.frame("country" = rep(c("France", "Spain", "Germany"), each = 10),
  "capital" = rep(c("Paris", "Madrid", "Berlin"), each = 10),
  "year" = 2009:2018,
  "gdp" = c(NA, NA, 200, 300, 500, 1000, NA, NA, NA, 500, 0, NA, NA, NA, NA, NA, 800, 1200, 1500, 100, 200, 400, 700, 700, 800, 600, 500, NA, NA))
jeu_donnees <- na.omit(jeu_donnees)# we artificially create some gaps in the time series
data_1 <- create_calendar(data = jeu_donnees, key_variable = "country", time_variable = "year",
  start_year = 2009, end_year = 2018)
data_2 <- start_end_to_fill(data = jeu_donnees, calendar = data_1, gap_variable = "gdp",
  key_variable = "country", time_variable = "year")
data_3 <- gap_to_fill(data = data_2, gap_variable = "gdp_corrected_1", key_variable = "country",
  time_variable = "year", digits = 1)
```

---

kmeans_procedure

**Kmeans procedure**

**Description**

This function allows to perform k-means clustering with constrained on the size of clusters

**Usage**

```r
kmeans_procedure(
  data,
  columns,
  threshold_min,
  threshold_max,
  verbose = FALSE,
  seed = 42
)
```

**Arguments**

- **data** R data frame
- **columns** vector of columns names of the data frame on which we perform the kmeans algorithm. These features have to be numeric.
- **threshold_min** integer that represents the minimum size for cluster.
- **threshold_max** integer that represents the maximum size for cluster.
- **verbose** boolean. if set to TRUE displays the current state of the procedure (by default set to FALSE).
- **seed** integer that represents seed for the random call (if we want the output to be reproducible).
**Value**

a R data frame contains the number of observation in the original data frame and a column `cluster` representing the cluster to which the observation belongs to.

**Author(s)**

Simon CORDE

**References**

Link to the github package repository: [https://www.github.com/Redcart/helda](https://www.github.com/Redcart/helda)

**Examples**

```r
data <- iris[, c(1:4)]
features <- colnames(iris)[c(1:4)]
result <- kmeans_procedure(data=data, columns=features, threshold_min=2, threshold=10, verbose=FALSE, seed=10)
```

---

**Description**

This function allows to draw a lift curve in a ggplot style for binary classification model

**Usage**

```r
lift_curve(predictions, true_labels, positive_label)
```

**Arguments**

- `predictions` vector of predictions. These are generally the result of a machine learning model. The predictions must be probabilities (a real number between 0 and 1).
- `true_labels` vector of true labels.
- `positive_label` string that specify the positive label (Y=1) in the `true_labels`

**Value**

a ggplot object containing the lift curve

**Author(s)**

Simon CORDE
lift_effect

References

Link to the author’s github package repository: https://www.github.com/Redcart/helda

Examples

data_training <- titanic_training
data_validation <- titanic_validation
model_glm <- glm(formula="Survived ~ Pclass + Sex + Age + SibSp + Fare + Embarked",
data=data_training,
family=binomial(link="logit"))
predictions <- predict(object=model_glm, newdata=titanic_validation, type="response")
lift_curve(predictions=predictions, true_labels=titanic_validation$Survived, positive_label=1)

---

**lift_effect**

**Lift effect curve**

**Description**

This function allows to draw the lift effect on a graph for binary classification model

**Usage**

lift_effect(predictions, true_labels, positive_label)

**Arguments**

predictions vector of predictions. These are generally the result of a machine learning model. The predictions must be probabilities (a real number between 0 and 1).

true_labels vector of true labels.

positive_label string that specify the positive label (Y=1) in the ‘true_labels’

**Value**

a ggplot object containing the lift effect

**Author(s)**

Simon CORDE

**References**

Link to the author’s github package repository: https://www.github.com/Redcart/helda
Examples

```r
data_training <- titanic_training
data_validation <- titanic_validation
model_glm <- glm(formula="Survived ~ Pclass + Sex + Age + SibSp + Fare + Embarked",
data=data_training,
family=binomial(link="logit"))
predictions <- predict(object=model_glm, newdata=titanic_validation, type="response")
lift_effect(predictions=predictions, true_labels=titanic_validation$Survived, positive_label=1)
```

---

**proc_freq**

*SAS proc freq in R*

---

**Description**

This function permits to reproduce the output of the SAS proc freq

**Usage**

```r
proc_freq(variable, digits = 4)
```

**Arguments**

- `variable`: vector on which we want to apply the function
- `digits`: integer that specifies the number of decimals we want to keep in the rounded figures

**Value**

A R data frame of dimension [number of categories x 5]. The five columns display the following information:

- Category: different categories of the original categorical variable
- Frequency
- Percentage
- Cumulative.Frequency
- Cumulative.Percentage

**Author(s)**

Simon CORDE

**References**

Link to the author’s github package repository: [https://www.github.com/Redcart/helda](https://www.github.com/Redcart/helda)
Examples

data <- iris
str(data)
result <- proc_freq(data$Species)

Description

This function allows to fill the start and end gaps of a time series by doing repetition of next (for the start) and previous values (for the end).

Usage

start_end_to_fill(data, calendar, gap_variable, key_variable, time_variable)

Arguments

- data: R data frame
- calendar: R data frame complete empty calendar (as one can perform with create_calendar_day)
- gap_variable: character that represents name of the variable we want to fill the start and end gaps
- key_variable: character that represents variable name that refers to the key variable in the panel data (ID, ...)
- time_variable: character that represents time variable name that permits to sort observation on a time scale

Value

- a R data frame containing the original columns and a new one:
  - gap_variable_corrected_1: the gap variable with starts and ends filled

Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://www.github.com/Redcart/helda
Examples

rep(c("Paris", "Madrid", "Berlin"), each = 10)
jeu_donnees <- data.frame("country" = rep(c("France", "Spain", "Germany"), each = 10),
"capital" = rep(c("Paris", "Madrid", "Berlin"), each = 10),
"year" = 2009:2018,
"gdp" = c(NA, NA, 200, 300, 500, 1000, NA, NA, NA, 500,
0, NA, NA, NA, NA, NA, 800, 1200, 1500,
100, 200, 400, 700, 700, 800, 600, 500, NA, NA))
jeu_donnees <- na.omit(jeu_donnees)# we artificially create some gaps in the time series
data_1 <- create_calendar(data = jeu_donnees, key_variable = "country", time_variable = "year",
start_year = 2009, end_year = 2018)
data_2 <- start_end_to_fill(data = jeu_donnees, calendar = data_1, gap_variable = "gdp",
key_variable = "country", time_variable = "year")
data_3 <- gap_to_fill(data = data_2, gap_variable = "gdp.corrected_1", key_variable = "country",
time_variable = "year", digits = 1)

titanic_testing

Titanic testing data set

Usage
titanic_testing

Format

Data frame of 418 rows and 10 columns

Examples
data(titanic_testing)

titanic_training

Titanic training data set

Description

Titanic training data set

Usage
titanic_training
**titanic_validation**

**Format**
- Data frame of 712 rows and 12 columns

**Examples**
```r
data(titanic_training)
```

---

**windows_to_linux_path**  
*Convert windows path into linux path*

**Description**
- This function allows to make conversion of windows path into linux path

**Usage**
```r
windows_to_linux_path()
```

**Details**
- When the function is called, a prompt asks for the windows path to be converted in the R console. Enter a windows path or copy paste one. Then type ENTER. The Linux converted path appears.

**Value**
- None
Author(s)

Simon CORDE

References

Link to the author’s github package repository: https://www.github.com/Redcart/helda

---

world_countries_pop  World countries population from 1960 to 2018

Description

World countries population from 1960 to 2018

Usage

world_countries_pop

Format

R Data frame with columns

- **country_name**  name of the country
- **country_code**  code of the country in three letters
- **year**  year from 1960 to 2018
- **population**  number of people

Examples

data(world_countries_pop)
Index

*Topic SAS
  proc_freq, 10
*Topic agglomerative
  compute_inertia_ahc, 3
*Topic calendar
  create_calendar, 4
*Topic categorical
  proc_freq, 10
*Topic centroids
  clust_centroid, 2
*Topic classification
  lift_curve, 8
  lift_effect, 9
*Topic clustering
  compute_inertia_ahc, 3
*Topic cluster
  kmeans_procedure, 7
*Topic curve
  lift_curve, 8
  lift_effect, 9
*Topic datasets
  titanic_testing, 12
  titanic_training, 12
  titanic_validation, 13
  world_countries_pop, 14
*Topic data
  clust_centroid, 2
  compute_inertia, 3
  compute_inertia_ahc, 3
  create_formula, 5
*Topic fill
  create_calendar, 4
  gap_to_fill, 6
  start_end_to_fill, 11
*Topic formula
  create_formula, 5
*Topic frame
  clust_centroid, 2
  compute_inertia, 3
  compute_inertia_ahc, 3
*Topic interpolation
  gap_to_fill, 6
*Topic kmeans
  kmeans_procedure, 7
*Topic learning
  lift_curve, 8
  lift_effect, 9
*Topic lift
  lift_curve, 8
  lift_effect, 9
*Topic linux
  windows_to_linux_path, 13
*Topic machine
  lift_curve, 8
  lift_effect, 9
*Topic path
  windows_to_linux_path, 13
*Topic proc_freq
  proc_freq, 10
*Topic series
  create_calendar, 4
  gap_to_fill, 6
  start_end_to_fill, 11
*Topic sizes
  kmeans_procedure, 7
*Topic table
  proc_freq, 10
*Topic time
create_calendar, 4
  gap_to_fill, 6
  start_end_to_fill, 11
*Topic variable
  proc_freq, 10
*Topic windows
  windows_to_linux_path, 13

clust_centroid, 2
compute_inertia, 3
compute_inertia_ahc, 3
create_calendar, 4
create_formula, 5

gap_to_fill, 6

kmeans_procedure, 7

lift_curve, 8
lift_effect, 9

proc_freq, 10

start_end_to_fill, 11

titanic_testing, 12
titanic_training, 12
titanic_validation, 13

windows_to_linux_path, 13
world_countries_pop, 14