

Package ‘kibior’

October 13, 2022

Type Package

Date 2021-01-27

Version 0.1.1

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.1.1

URL <https://github.com/regisoc/kibior>

BugReports <https://github.com/regisoc/kibior/issues>

Title A Simple Data Management and Sharing Tool

biocViews DataImport, DataRepresentation, ThirdPartyClient

Description An interface to store, retrieve, search, join and share datasets, based on Elastic-search (ES) API. As a decentralized, FAIR and collaborative search engine and database effort, it proposes a simple push/pull/search mechanism only based on ES, a tool which can be deployed on nearly any hardware. It is a high-level R-ES binding to ease data usage using 'elastic' package (S. Chamberlain (2020)) <<https://docs.ropensci.org/elastic/>>, extends joins from 'dplyr' package (H. Wickham et al. (2020)) <<https://dplyr.tidyverse.org/>> and integrates specific biological format importation with Bioconductor packages such as 'rtracklayer' (M. Lawrence and al. (2009) <[doi:10.1093/bioinformatics/btp328](https://doi.org/10.1093/bioinformatics/btp328)>) <<http://bioconductor.org/packages/rtracklayer>>, 'Biostrings' (H. Pagès and al. (2020) <[doi:10.18129/B9.bioc.Biostrings](https://doi.org/10.18129/B9.bioc.Biostrings)>) <<http://bioconductor.org/packages/Biostrings>>, and 'Rsamtools' (M. Morgan and al. (2020) <[doi:10.18129/B9.bioc.Rsamtools](https://doi.org/10.18129/B9.bioc.Rsamtools)>) <<http://bioconductor.org/packages/Rsamtools>>, but also a long list of more common ones with 'rio' (C-h. Chan and al. (2018)) <<https://cran.r-project.org/package=rio>>.

Depends R (>= 4.0)

Imports R6 (>= 2.5.0), data.table (>= 1.13.2), stringr (>= 1.4.0), purrr (>= 0.3.4), jsonlite (>= 1.7.1), rio (>= 0.5.16), tibble (>= 3.0.4), dplyr (>= 1.0.2), magrittr (>= 1.5), tidyr (>= 1.1.2), elastic (>= 1.1.0), Biostrings, Rsamtools, rtracklayer

Suggests ggplot2 (>= 3.3.2), readr (>= 1.4.0), xml2 (>= 1.3.2), yaml (>= 2.2.1), testthat (>= 3.0.0), rmarkdown (>= 2.5), knitr (>= 1.30)

License GPL-2

LazyData true

NeedsCompilation no

Author Régis Ongaro-Carcy [aut, cre, ctb]
(<<https://orcid.org/0000-0002-0091-8810>>),
Arnaud Droit [fnd]

Maintainer Régis Ongaro-Carcy <regis.ongaro-carcy2@crchudequebec.ulaval.ca>

Repository CRAN

Date/Publication 2021-01-28 15:20:02 UTC

R topics documented:

kibior	2
Kibior equals operator	53
Kibior not-equals operator	54
Static - initiate a direct instance to Kibio public repository	55
Static - Kibior is instance	55
Static - Tests if packages are installed	56
Index	57

kibior

Kibior, an Kibio and Elasticsearch data manipulation package.

Description

Kibior is a lightweight package for data manipulation with Elasticsearch. Its main features allow easy data import, export, download, upload, searching and sharing to any Elasticsearch-based open architecture, scaling to billions of data and TB capability.

Kibior is a Kibio/Elasticsearch client written with R6 class. Instances of Kibior are object that allow to use Elasticsearch power and execute lots of predefined requests such as searching in massive amounts of data, joins between in-memory data and Elasticsearch indices, push and pull data to and from multiples Elasticsearch servers, and more. This little utility was built in the context of massive data invading biology and bioinformatics, but is completely versatile and can be applied to other fields. By adding it to R-scripts, it can perform several useful tasks such as: saving intermediary results, sharing them with a collaborator, automating import and upload of lots of files directly, and much more.

Format

R6Class object.

Details

A client to send, retrieve, search, join data in Elasticsearch.

Constructor Arguments

Argument	Type	Details	Default
host	character	address or name of Elasticsearch server	"localhost"
port	numeric	port of Elasticsearch server	9200
user	character	if required by the server, the username for authentication	NULL
pwd	character	if required by the server, the password for authentication	NULL
verbose	logical	verbose mode	FALSE

created

Public fields

verbose verbose mode, prints out more informations during execution

quiet_progress progressbar quiet mode, toggles progress bar

quiet_results results quiet mode, toggles results printing

Active bindings

host Access and change the Elasticsearch host

port Access and change the Elasticsearch port

endpoint Access the Elasticsearch main endpoint

user Access the Elasticsearch user.

pwd Access the Elasticsearch password.

connection Access the Elasticsearch connection object.

head_search_size Access and change the head size default value.

cluster_name Access the cluster name if and only if already connected.

cluster_status Access the cluster status if and only if already connected.

nb_documents Access the current cluster total number of documents if and only if already connected.

version Access the Elasticsearch version if and only if already connected.

elastic_wait Access and change the Elasticsearch wait time for update commands if and only if already connected.

valid_joins Access the valid joins available in Kibior.

valid_count_types Access the valid count types available (mainly observations = rows, variables = columns)

valid_elastic_metadata_types Access the valid Elasticsearch metadata types available.

valid_push_modes Access the valid push modes available.

`shard_number` Access and modify the number of allocated primary shards when creating an Elasticsearch index.

`shard_replicas_number` Access and modify the number of allocated replicas in an Elasticsearch index.

`default_id_col` Access and modify the default ID column/field created when pushing data to Elasticsearch.

Methods

Public methods:

- `Kibior$new()`
- `Kibior$print()`
- `Kibior$eq()`
- `Kibior$ne()`
- `Kibior$create()`
- `Kibior$list()`
- `Kibior$has()`
- `Kibior$delete()`
- `Kibior$add_description()`
- `Kibior$has_description()`
- `Kibior$missing_descriptions()`
- `Kibior$remove_description()`
- `Kibior$clean_descriptions()`
- `Kibior$describe()`
- `Kibior$describe_index()`
- `Kibior$describe_columns()`
- `Kibior$infos()`
- `Kibior$ping()`
- `Kibior$mappings()`
- `Kibior$settings()`
- `Kibior$aliases()`
- `Kibior$dim()`
- `Kibior$columns()`
- `Kibior$count()`
- `Kibior$avg()`
- `Kibior$mean()`
- `Kibior$min()`
- `Kibior$max()`
- `Kibior$sum()`
- `Kibior$stats()`
- `Kibior$percentiles()`
- `Kibior$q1()`
- `Kibior$q2()`

- `Kibior$median()`
- `Kibior$q3()`
- `Kibior$summary()`
- `Kibior$keys()`
- `Kibior$bam_to_tibble()`
- `Kibior$soft_cast()`
- `Kibior$get_resource()`
- `Kibior$export()`
- `Kibior$import_tabular()`
- `Kibior$import_features()`
- `Kibior$import_alignments()`
- `Kibior$import_json()`
- `Kibior$import_sequences()`
- `Kibior$guess_import()`
- `Kibior$import()`
- `Kibior$push()`
- `Kibior$pull()`
- `Kibior$move()`
- `Kibior$copy()`
- `Kibior$match()`
- `Kibior$search()`
- `Kibior$inner_join()`
- `Kibior$full_join()`
- `Kibior$left_join()`
- `Kibior$right_join()`
- `Kibior$semi_join()`
- `Kibior$anti_join()`
- `Kibior$clone()`

Method `new()`:

Usage:

```
Kibior$new(  
  host = "localhost",  
  port = 9200,  
  user = NULL,  
  pwd = NULL,  
  verbose = getOption("verbose")  
)
```

Arguments:

`host` The target host to connect to Elasticsearch REST API (default: "localhost").

`port` The target port (default: 9200).

`user` If the server needs authentication, your username (default: NULL).

`pwd` If the server needs authentication, your password (default: NULL).

verbose The verbose mode (default: FALSE).

Details: Initialize a new object, automatically called when calling ‘Kibior\$new()’

Returns: a new instance/object of Kibior

Examples:

```
\dontrun{
# default initialization, connect to "localhost:9200"
kc <- Kibior$new()
# connect to "192.168.2.145:9200"
kc <- Kibior$new("192.168.2.145")
# connect to "es:15005", verbose mode activated
kc <- Kibior$new(host = "elasticsearch", port = 15005, verbose = TRUE)
# connect to "192.168.2.145:9450" with credentials "foo:bar"
kc <- Kibior$new(host = "192.168.2.145", port = 9450, user = "foo", pwd = "bar")
# connect to "elasticsearch:9200"
kc <- Kibior$new("elasticsearch")

# get kibior var from env (".Renvirom" file or local env)
dd <- system.file("doc_env", "kibior_build.R", package = "kibior")
source(dd, local = TRUE)
kc <- .kibior_get_instance_from_env()
kc$quiet_progress <- TRUE

# preparing all examples (do not mind this for this method)
delete_if_exists <- function(index_names){
  tryCatch(
    expr = { kc$delete(index_names) },
    error = function(e){ }
  )
}
delete_if_exists(c(
  "aaa",
  "bbb",
  "ccc",
  "ddd",
  "sw",
  "sw_naboo",
  "sw_tatooine",
  "sw_alderaan",
  "sw_from_file",
  "storms",
  "starwars"
))
}
```

Method print():

Usage:

```
Kibior$print()
```

Details: Print simple informations of the current object.

Examples:

```
\dontrun{  
print(kc)  
}
```

Method eq():

Usage:

```
Kibior$eq(other = NULL)
```

Arguments:

other Another instance/object of Kibior (default: NULL).

Details: Tells if another instance of Kibior has the same 'host:port' couple.

Returns: TRUE if hosts and ports are identical, else FALSE

Examples:

```
\dontrun{  
kc$eq(kc)  
}
```

Method ne():

Usage:

```
Kibior$ne(other = NULL)
```

Arguments:

other Another instance/object of Kibior (default: NULL).

Details: Tells if another instance of Kibior has a different 'host:port' couple.

Returns: TRUE if hosts and ports are differents, else FALSE

Examples:

```
\dontrun{  
kc$ne(kc)  
}
```

Method create():

Usage:

```
Kibior$create(index_name, force = FALSE)
```

Arguments:

index_name a vector of index names to create (default: NULL).

force Erase already existing identical index names? (default: FALSE).

Details: Create one or several indices in Elasticsearch.

Returns: a list containing results of creation per index

Examples:

```
\dontrun{
kc$create("aaa")
kc$create(c("bbb", "ccc"))
}
```

Method list():

Usage:

```
Kibior$list(get_specials = FALSE)
```

Arguments:

`get_specials` a boolean to get special indices (default: FALSE).

Details: List indices in Elasticsearch.

Returns: a list of index names, NULL if no index found

Examples:

```
\dontrun{
kc$list()
kc$list(get_specials = TRUE)
}
```

Method has():

Usage:

```
Kibior$has(index_name)
```

Arguments:

`index_name` a vector of index names to check.

Details: Does Elasticsearch has one or several indices?

Returns: a list with TRUE for found index, else FALSE

Examples:

```
\dontrun{
kc$has("aaa")
kc$has(c("bbb", "ccc"))
}
```

Method delete():

Usage:

```
Kibior$delete(index_name)
```

Arguments:

`index_name` a vector of index names to delete.

Details: Delete one or several indices in Elasticsearch.

Returns: a list containing results of deletion per index, or NULL if no index name match

Examples:

```
\dontrun{
kc$delete("aaa")
kc$delete(c("bbb", "ccc"))
}
```

Method add_description():

Usage:

```
Kibior$add_description(
  index_name,
  dataset_name,
  source_name,
  index_description,
  version,
  change_log,
  website,
  direct_download,
  version_date,
  license,
  contact,
  references,
  columns = list(),
  force = FALSE
)
```

Arguments:

index_name the index name to describe

dataset_name the full length dataset name

source_name the source/website/entity full length name

index_description the index description, should be explicit

version the version of the source dataset

change_log what have been done from the last version

website the website to the source dataset website

direct_download the direct download url of the dataset source

version_date the version or build date

license the license attached to this dataset, could be a url

contact a mailto/contact

references some paper and other references (e.g. doi, url)

columns a list of (column_name = column_description) to register (default: list())

force if FALSE, raise an error if the description already exists, else TRUE to overwrite (default: FALSE)

Details: Add a description of a pushed dataset.

Returns: the index name if complete, else an error

Examples:

```
\dontrun{
kc$add_description(
  index_name = "sw",
  dataset_name = "starwars",
  source_name = "Package dplyr",
  index_description = "Description of starwars characters, the data comes from the Star Wars API.",
  version = "dplyr (1.0.0)",
  link = "http://swapi.dev/",
  direct_download_link = "http://swapi.dev/",
  version_date = "2020-05-28",
  license_link = "MIT",
  columns = list(
    "name" = "Name of the character",
    "height" = "Height (cm)",
    "mass" = "Weight (kg)",
    "hair_color" = "Hair colors",
    "skin_color" = "Skin colors",
    "eye_color" = "Eye colors",
    "birth_year" = "Year born (BBY = Before Battle of Yavin)",
    "sex" = "The biological sex of the character, namely male, female, hermaphroditic, or none (as in the case for Droids).",
    "gender" = "The gender role or gender identity of the character as determined by their personality or the way they were programmed (as in the case for Droids).",
    "homeworld" = "Name of homeworld",
    "species" = "Name of species",
    "films" = "List of films the character appeared in",
    "vehicles" = "List of vehicles the character has piloted",
    "starships" = "List of starships the character has piloted"
  )
)
}
```

Method has_description():

Usage:

```
Kibior$has_description(index_name)
```

Arguments:

index_name the index name to describe

Details: Does the description exists?

Returns: a list splitted by index, with TRUE if the description is found, else FALSE. Removes unknown index names.

Examples:

```
\dontrun{
kc$has_description("s*")
kc$has_description(c("sw", "asdf"))
}
```

Method missing_descriptions():

Usage:

```
Kibior$missing_descriptions()
```

Details: List indices that do not have descriptions.

Returns: a vector of indices not present in description.

Examples:

```
\dontrun{
kc$missing_descriptions()
}
```

Method remove_description():

Usage:

```
Kibior$remove_description(index_name)
```

Arguments:

index_name the index name to describe

Details: Remove a description.

Returns: a vector of indices not present in description.

Examples:

```
\dontrun{
# remove the description of 'test' index
kc$remove_description("test")
}
```

Method clean_descriptions():

Usage:

```
Kibior$clean_descriptions()
```

Details: Remove all descriptions that do not have an index associated.

Returns: a list of index names which have been removed from descriptions.

Examples:

```
\dontrun{
# remove the description of 'test' index
kc$clean_descriptions()
}
```

Method describe():*Usage:*

```
Kibior$describe(index_name, columns = NULL, pretty = FALSE)
```

Arguments:

`index_name` the index name to describe

`columns` a vector of column names to describe (default: NULL)

`pretty` pretty-print the result (default: FALSE)

Details: Get the description of indices and their columns.

Returns: all description, grouped by indices

Examples:

```
\dontrun{
kc$describe("s*")
kc$describe("sw", columns = c("name", "height"))
}
```

Method describe_index():*Usage:*

```
Kibior$describe_index(index_name)
```

Arguments:

`index_name` the index name to describe

Details: Get the description text of indices.

Returns: a list of description text, grouped by indices

Examples:

```
\dontrun{
kc$describe_index("s*")
}
```

Method describe_columns():*Usage:*

```
Kibior$describe_columns(index_name, columns)
```

Arguments:

`index_name` the index name to describe

`columns` a vector of column names to describe

Details: Get the description text of index columns.

Returns: a list of description text, grouped by indices

Examples:

```
\dontrun{
kc$describe_columns("s*", c("name", "height"))
}
```

Method infos():

Usage:

```
Kibior$infos()
```

Details: Get informations about Elasticsearch cluster

Returns: a list of statistics about the cluster

Examples:

```
\dontrun{  
kc$infos()  
}
```

Method ping():

Usage:

```
Kibior$ping()
```

Details: Ping cluster connection

Returns: the ping result with some basic infos

Examples:

```
\dontrun{  
kc$ping()  
}
```

Method mappings():

Usage:

```
Kibior$mappings(index_name)
```

Arguments:

index_name a vector of index names to get mappings.

Details: Get mappings of indices

Returns: the list of indices, containing their mapping

Examples:

```
\dontrun{  
kc$mappings()  
kc$mappings("sw")  
kc$mappings(c("sw", "sw_naboo"))  
}
```

Method settings():

Usage:

```
Kibior$settings(index_name)
```

Arguments:

index_name a vector of index names to get settings.

Details: Get settings of indices

Returns: the list of indices, containing their settings

Examples:

```
\dontrun{
kc$settings()
kc$settings("sw")
kc$settings(c("sw", "sw_tatooine"))
}
```

Method aliases():

Usage:

```
Kibior$aliases(index_name)
```

Arguments:

`index_name` a vector of index names to get aliases.

Details: Get aliases of indices

Returns: the list of indices, containing their aliases

Examples:

```
\dontrun{
kc$aliases()
kc$aliases("sw")
kc$aliases(c("sw", "sw_alderaan"))
}
```

Method dim():

Usage:

```
Kibior$dim(index_name)
```

Arguments:

`index_name` a vector of index names to get aliases.

Details: Shortcut to `$count()` to match the classical `'dim()'` function pattern `'[line col]'`

Returns: the list of indices, containing their number of observations and variables.

Examples:

```
\dontrun{
# Couple [<nb obs> <nb var>] in "sw"
kc$dim("sw")
# Couple [<nb obs> <nb var>] in indices "sw_naboo" and "sw_alderaan"
kc$dim(c("sw_naboo", "sw_alderaan"))
}
```

Method columns():

Usage:

```
Kibior$columns(index_name)
```

Arguments:

`index_name` a vector of index names, can be a pattern.

Details: Get fields/columns of indices.

Returns: a list of indices, each containing their fields/columns.

Examples:

```
\dontrun{
kc$columns("sw")           # direct search
kc$columns("sw_*")        # pattern search
}
```

Method count():

Usage:

```
Kibior$count(index_name, type = "observations", query = NULL)
```

Arguments:

`index_name` a vector of index names to get aliases.

`type` a string representing the type to count: "observations" (lines) or "variables" (columns) (default: "observations").

`query` a string as a query string syntax (default: NULL).

Details: Count observations or variables in Elasticsearch data

Returns: the list of indices, containing their number of observations or variables. Use `$dim()` for both

Examples:

```
\dontrun{
# Number of observations (nb of records) in "sw"
kc$count("sw")
# Number of observations in indices "sw_naboo" and "sw_tatooine"
kc$count(c("sw_naboo", "sw_tatooine"))
# Number of variables (nb of columns) in index "sw_naboo"
kc$count("sw_naboo", type = "variables")
}
```

Method avg():

Usage:

```
Kibior$avg(index_name, columns, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`query` a string as a query string syntax (default: NULL).

Details: Get the average of numeric columns.

Returns: a tibble with avg, one line by matching index and column.

Examples:

```
\dontrun{
# Avg of "sw" column "height"
kc$avg("sw", "height")
# if pattern
kc$avg("s*", "height")
# multiple indices, multiple columns
kc$avg(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method mean():

Usage:

```
Kibior$mean(index_name, columns, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`query` a string as a query string syntax (default: NULL).

Details: Get the mean of numeric columns.

Returns: a tibble with mean, one line by matching index and column.

Examples:

```
\dontrun{
# mean of "sw" column "height"
kc$mean("sw", "height")
# if pattern
kc$mean("s*", "height")
# multiple indices, multiple columns
kc$mean(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method min():

Usage:

```
Kibior$min(index_name, columns, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`query` a string as a query string syntax (default: NULL).

Details: Get the minimum of numeric columns.

Returns: a tibble with min, one line by matching index and column.

Examples:


```

\dontrun{
# min of "sw" column "height"
kc$min("sw", "height")
# if pattern
kc$min("s*", "height")
# multiple indices, multiple columns
kc$min(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}

```

Method max():*Usage:*

```
Kibior$max(index_name, columns, query = NULL)
```

Arguments:

index_name a vector of index names.

columns a vector of column names.

query a string as a query string syntax (default: NULL).

Details: Get the maximum of numeric columns.

Returns: a tibble with max, one line by matching index and column.

Examples:

```

\dontrun{
# max of "sw" column "height"
kc$max("sw", "height")
# if pattern
kc$max("s*", "height")
# multiple indices, multiple columns
kc$max(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}

```

Method sum():*Usage:*

```
Kibior$sum(index_name, columns, query = NULL)
```

Arguments:

index_name a vector of index names.

columns a vector of column names.

query a string as a query string syntax (default: NULL).

Details: Get the sum of numeric columns.

Returns: a tibble with sum, one line by matching index and column.

Examples:

```

\dontrun{
# sum of "sw" column "height"
kc$sum("sw", "height")
}

```

```
# if pattern
kc$sum("s*", "height")
# multiple indices, multiple columns
kc$sum(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method stats():*Usage:*

```
Kibior$stats(index_name, columns, sigma = NULL, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`sigma` (default: NULL).

`query` a string as a query string syntax (default: NULL).

Details: Produces descriptive statistics of a column. Returns a tibble composed of: count, min, max, avg, sum, sum_of_squares, variance, std_deviation (+ upper and lower bounds). Multiple warnings here. One for the count and one for the std_dev. 1/ Counts: they are approximate, see vignette. 2/ Std dev: as stated in ES documentation: "The standard deviation and its bounds are displayed by default, but they are not always applicable to all data-sets. Your data must be normally distributed for the metrics to make sense. The statistics behind standard deviations assumes normally distributed data, so if your data is skewed heavily left or right, the value returned will be misleading."

Returns: a tibble with descriptive stats, one line by matching index.

Examples:

```
\dontrun{
# Stats of "sw" column "height"
kc$stats("sw", "height")
# if pattern
kc$stats("s*", "height")
# multiple indices and sigma definition
kc$stats(c("sw", "sw2"), "height", sigma = 2.5)
# multiple indices, multiple columns
kc$stats(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method percentiles():*Usage:*

```
Kibior$percentiles(index_name, columns, percents = NULL, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`percents` a numeric vector of percents to use (default: NULL).

`query` a string as a query string syntax (default: NULL).

Details: Get percentiles of numeric columns.

Returns: a list of tibble, splitted by indices with percentiles inside tibble columns.

Examples:

```
\dontrun{
# percentiles of "sw" column "height", default is with q1, q2 and q3
kc$percentiles("sw", "height")
# if pattern
kc$percentiles("s*", "height")
# defining percents to get
kc$percentiles("s*", "height", percents = c(20, 25))
# multiple indices, multiple columns
kc$percentiles(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method q1():

Usage:

```
Kibior$q1(index_name, columns, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`query` a string as a query string syntax (default: NULL).

Details: Get Q1 percentiles from numeric columns.

Returns: a list of tibble, splitted by indices with Q1 inside tibble columns.

Examples:

```
\dontrun{
# Q1 of "sw" column "height"
kc$q1("sw", "height")
# if pattern
kc$q1("s*", "height")
# multiple indices, multiple columns
kc$q1(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method q2():

Usage:

```
Kibior$q2(index_name, columns, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`query` a string as a query string syntax (default: NULL).

Details: Get Q2 percentiles from numeric columns.

Returns: a list of tibble, splitted by indices with Q2 inside tibble columns.

Examples:

```
\dontrun{
# Q2 of "sw" column "height"
kc$q2("sw", "height")
# if pattern
kc$q2("s*", "height")
# multiple indices, multiple columns
kc$q2(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method median():

Usage:

```
Kibior$median(index_name, columns, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`query` a string as a query string syntax (default: NULL).

Details: Get median from numeric columns. Basically a wrapper around ‘\$q2()’.

Returns: a list of tibble, splitted by indices with median inside tibble columns.

Examples:

```
\dontrun{
# median of "sw" column "height"
kc$median("sw", "height")
# if pattern
kc$median("s*", "height")
# multiple indices, multiple columns
kc$median(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}
```

Method q3():

Usage:

```
Kibior$q3(index_name, columns, query = NULL)
```

Arguments:

`index_name` a vector of index names.

`columns` a vector of column names.

`query` a string as a query string syntax (default: NULL).

Details: Get Q3 percentiles from numeric columns.

Returns: a list of tibble, splitted by indices with Q3 inside tibble columns.

Examples:

```

\dontrun{
# Q3 of "sw" column "height"
kc$q3("sw", "height")
# if pattern
kc$q3("s*", "height")
# multiple indices, multiple columns
kc$q3(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}

```

Method summary():*Usage:*

Kibior\$summary(index_name, columns, query = NULL)

Arguments:

index_name a vector of index names.

columns a vector of column names.

query a string as a query string syntax (default: NULL).

Details: Summary for numeric columns. Cumulates '\$min()', '\$max()', '\$q1()', '\$q2()', '\$q3()'.

Returns: a list of tibble, splitted by indices.

Examples:

```

\dontrun{
# summary of "sw" column "height"
kc$summary("sw", "height")
# if pattern
kc$summary("s*", "height")
# multiple indices, multiple columns
kc$summary(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")
}

```

Method keys():*Usage:*

Kibior\$keys(index_name, column, max_size = 1000)

Arguments:

index_name an index name.

column a field name of this index (default: NULL).

max_size the maximum result to return (default: 1000).

Details: Get distinct keys elements of a specific column.

Returns: a vector of keys values from this field/column

Examples:

```

\dontrun{
kc$keys("sw", "name")
kc$keys("sw", "eye_color")
}

```

Method bam_to_tibble():*Usage:*

Kibior\$bam_to_tibble(bam_data = NULL)

Arguments:

bam_data data from a BAM file (default: NULL).

Details: Transformation function for collapsing the BAM list of lists format into a single list as per the Rsamtools vignette*Returns:* a tibble of BAM data*Examples:*

```
\dontrun{
dd_bai <- system.file("extdata", "test.bam.bai", package = "kibior")
bam_param <- Rsamtools::ScanBamParam(what = c("pos", "qwidth"))
bam_data <- Rsamtools::scanBam(dd_bai, param = bam_param)
kc$bam_to_tibble(bam_data)
}
```

Method soft_cast():*Usage:*

```
Kibior$soft_cast(
  data,
  caster = getFromNamespace("as_tibble", "tibble"),
  caster_args = list(.name_repair = "unique"),
  warn = TRUE
)
```

Arguments:

data data to cast.

caster the caster closure/function (default: tibble::as_tibble)

caster_args others caster args (default: list(.name_repair = "unique"))

warn do print warning if error? (default: TRUE)

Details: Casting function that tries to cast a transformation closure. Uses tibble::as_tibble() by default.*Returns:* a cast or the unchanged data.*Examples:*

```
\dontrun{
kc$soft_cast(datasets::iris)
}
```

Method get_resource():*Usage:*

Kibior\$get_resource(url_or_filepath, fileext = NULL)

Arguments:

`url_or_filepath` a filepath or an URL.
`fileext` the file extension (default: NULL).

Details: Get a local filepath or an URL data through a tempfile. If the file exists locally, the filepath will be returned, if not, it will tries to download the data and return the temp filepath.

Returns: a filepath.

Examples:

```
\dontrun{
kc$get_resource(system.file("R", "kibior.R", package = "kibior"))
kc$get_resource("https://ftp.ncbi.nlm.nih.gov/entrez/README")
}
```

Method `export()`:

Usage:

```
Kibior$export(data, filepath, format = "csv", force = FALSE)
```

Arguments:

`data` an index name or in-memory data to be extracted to a file.
`filepath` the filepath to use as export, must contain the file extention.
`format` the file format to use (default: "csv").
`force` overwrite the file? (default: FALSE).

Details: Export data to a file. Needs 'rio' package from CRAN. Some data formats are not installed by default. Use 'rio::install_formats()' to be able to parse them.

Returns: the filepath if correctly exported, else an error

Examples:

```
\dontrun{
f <- tempfile(fileext=".csv")
# export and overwrite last file with the same data from Elasticsearch
kc$export(data = "sw", filepath = f)
# export from in-memory data to a file
kc$export(data = dplyr::starwars, filepath = f, force = TRUE)
}
```

Method `import_tabular()`:

Usage:

```
Kibior$import_tabular(filepath, to_tibble = TRUE, fileext = ".csv")
```

Arguments:

`filepath` the filepath to use as import, must contain the file extention.
`to_tibble` returns the result as tibble? If FALSE, the raw default rio::import() format will be used (default: TRUE).
`fileext` the file extension (default: ".csv").

Details: Import method for tabular data. Needs 'rio' package from CRAN. Works mainly with CSV, TSV, TAB, TXT and ZIPped formats.

Returns: data contained in the file as a tibble, or NULL.

Examples:

```
\dontrun{
f <- tempfile(fileext = ".csv")
rio::export(ggplot2::diamonds, f)
# import to in-memory variable
kc$import_tabular(filepath = f)
# import raw data
kc$import_tabular(filepath = f, to_tibble = FALSE)
}
```

Method import_features():

Usage:

```
Kibior$import_features(filepath, to_tibble = TRUE, fileext = ".gtf")
```

Arguments:

filepath the filepath to use as import, must contain the file extension.

to_tibble returns the result as tibble? If FALSE, the raw default rtracklayer::import() format will be used (default: TRUE).

fileext the file extension (default: ".gtf").

Details: Import method for features data. Needs 'rtracklayer' package from Bioconductor. Works with BED, GTF, GFFx, and GZIPped formats.

Returns: data contained in the file as a tibble, or NULL.

Examples:

```
\dontrun{
# get sample files
f_gff <- system.file("extdata", "chr_y.gff3.gz", package = "kibior")
f_bed <- system.file("extdata", "cpg.bed", package = "kibior")
# import to in-memory variable
kc$import_features(filepath = f_bed)
kc$import_features(filepath = f_gff)
# import raw data
kc$import_features(filepath = f_bed, to_tibble = FALSE)
kc$import_features(filepath = f_gff, to_tibble = FALSE)
}
```

Method import_alignments():

Usage:

```
Kibior$import_alignments(filepath, to_tibble = TRUE, fileext = ".bam")
```

Arguments:

filepath the filepath to use as import, should contain the file extension.

to_tibble returns the result as tibble? If FALSE, the raw default Rsamtools::scanBam() format will be used (default: TRUE).

fileext the file extension (default: ".bam").

Details: Import method for alignments data. Needs 'Rsamtools' packages from Bioconductor. Works with BAM format.

Returns: data contained in the file as a tibble, or NULL.

Examples:

```
\dontrun{
# get sample file
f_bai <- system.file("extdata", "test.bam.bai", package = "kibior")
# import to in-memory variable
kc$import_alignments(filepath = f_bai)
# import raw data
kc$import_alignments(filepath = f_bai, to_tibble = FALSE)
}
```

Method import_json():

Usage:

```
Kibior$import_json(filepath, to_tibble = TRUE, fileext = ".json")
```

Arguments:

filepath the filepath to use as import, should contain the file extension.

to_tibble returns the result as tibble? If FALSE, the raw dataframe format will be used (default: TRUE).

fileext the file extension (default: ".json").

Details: Import method for JSON format. Needs 'jsonlite' packages from CRAN.

Returns: data contained in the file as a tibble, dataframe or NULL.

Examples:

```
\dontrun{
# get sample file
f_json <- system.file("extdata", "storms100.json", package = "kibior")
# import to in-memory variable
kc$import_json(f_json)
# import raw data
kc$import_json(f_json, to_tibble = FALSE)
}
```

Method import_sequences():

Usage:

```
Kibior$import_sequences(filepath, to_tibble = TRUE, fasta_type = "auto")
```

Arguments:

filepath the filepath to use as import, should contain the file extension.

to_tibble returns the result as tibble? If FALSE, the raw default Rsamtools::scanBam() format will be used (default: TRUE).

fasta_type type of parsing. It can be "dna", "rna", "aa" ou "auto" (default: "auto")

Details: Import method for sequences data. Needs 'Biostrings' package from Bioconductor. Works with FASTA formats.

Returns: data contained in the file as a tibble, or NULL.

Examples:

```
\dontrun{
# get sample file
f_dna <- system.file("extdata", "dna_human_y.fa.gz", package = "kibior")
f_rna <- system.file("extdata", "ncrna_mus_musculus.fa.gz", package = "kibior")
f_aa <- system.file("extdata", "pep_mus_spretus.fa.gz", package = "kibior")
# import to in-memory variable
kc$import_sequences(filepath = f_dna, fasta_type = "dna")
# import raw data
kc$import_sequences(filepath = f_rna, to_tibble = FALSE, fasta_type = "rna")
# import auto
kc$import_sequences(filepath = f_aa)
}
```

Method guess_import():

Usage:

```
Kibior$guess_import(filepath, to_tibble = TRUE)
```

Arguments:

filepath the filepath to use as import, must contain the file extension.

to_tibble returns the result as tibble? (default: TRUE).

Details: Import method that will try to guess importation method. Will also try to read from compressed data if they are. This method will call other import_* methods when trying. Some data formats are not installed by default. Use 'rio::install_formats()' to be able to parse them.

Returns: data contained in the file, or NULL.

Examples:

```
\dontrun{
# get sample file
f_dna <- system.file("extdata", "dna_human_y.fa.gz", package = "kibior")
f_rna <- system.file("extdata", "ncrna_mus_musculus.fa.gz", package = "kibior")
f_aa <- system.file("extdata", "pep_mus_spretus.fa.gz", package = "kibior")
f_bai <- system.file("extdata", "test.bam.bai", package = "kibior")
f_gff <- system.file("extdata", "chr_y.gff3.gz", package = "kibior")
f_bed <- system.file("extdata", "cpg.bed", package = "kibior")
# import
kc$guess_import(f_dna)
kc$guess_import(f_rna)
kc$guess_import(f_aa)
kc$guess_import(f_bai)
kc$guess_import(f_gff)
kc$guess_import(f_bed)
```

```
}

```

Method import():

Usage:

```
Kibior$import(
  filepath,
  import_type = "auto",
  push_index = NULL,
  push_mode = "check",
  id_col = NULL,
  to_tibble = TRUE
)
```

Arguments:

`filepath` the filepath to use as import, must contain the file extension.

`import_type` can be one of "auto", "tabular", "features", "alignments", "sequences" (default: "auto").

`push_index` the name of the index where to push data (default: NULL).

`push_mode` the push mode (default: "check").

`id_col` the column name of unique IDs (default: NULL).

`to_tibble` returns the result as tibble? (default: TRUE).

Details: Generic import method. This method will call other `import_*` methods when trying. Some data formats are not installed by default.

Returns: data contained in the file, or NULL.

Examples:

```
\dontrun{
# get sample file
f_aa <- system.file("extdata", "pep_mus_spretus.fa.gz", package = "kibior")
f_gff <- system.file("extdata", "chr_y.gff3.gz", package = "kibior")
f_bai <- system.file("extdata", "test.bam.bai", package = "kibior")
# import
kc$import(filepath = f_aa)
# import to Elasticsearch index ("sw_from_file") if not exists
kc$import(filepath = f_bai, push_index = "sw_from_file")
# import to index by recreating it, then pull indexed data
kc$import(filepath = f_gff, push_index = "sw_from_file",
  push_mode = "recreate")
}
```

Method push():

Usage:

```
Kibior$push(data, index_name, bulk_size = 1000, mode = "check", id_col = NULL)
```

Arguments:

`data` the data to push.

`index_name` the index name to use in Elasticsearch.

`bulk_size` the number of record to send to Elasticsearch in a row (default: 1000).

`mode` the push mode, could be "check", "recreate" or "update" (default: "check").

`id_col` an column anme to use as ID, must be composed of unique elements (default: NULL).

Details: Push data from in-memory to Elasticsearch. Everything is done by bulk.

Returns: the `index_name` given if the push ended well, else an error.

Examples:

```
\dontrun{
# erase the last push data by recreating the index and re-pushing data
kc$push(dplyr::starwars, index_name = "sw", mode = "recreate")
# characters names are unique, can be used as ID
kc$push(dplyr::starwars, index_name = "sw", mode = "recreate", id_col = "name")
# a bit more complicated: update some data of the dataset "starwars"
# 38 records on 87 filtered
some_new_data <- dplyr::filter(dplyr::starwars, height > 180)
# make them all "gender <- female"
some_new_data["gender"] <- "female"
# update that apply, based on cahracter names to match the right record
kc$push(some_new_data, "sw", mode = "update", id_col = "name")
# view result by querying
kc$pull("sw", query = "height:>180", columns = c("name", "gender"))
}
```

Method `pull()`:

Usage:

```
Kibior$pull(
  index_name,
  bulk_size = 500,
  max_size = NULL,
  scroll_timer = "3m",
  keep_metadata = FALSE,
  columns = NULL,
  query = NULL
)
```

Arguments:

`index_name` the index name to use in Elasticsearch.

`bulk_size` the number of record to send to Elasticsearch in a row (default: 500).

`max_size` the number of record Elasticsearch will send (default: NULL (all data)).

`scroll_timer` the time the scroll API will let the request alive to scroll on the result (default: "3m" (3 minute)).

`keep_metadata` does Elasticsearch needs to sent metadata? Data columns will be prefixed by "_source." (default: FALSE).

`columns` a vector of columns to select (default: NULL (all columns)).

query a string formatted to Elasticsearch query syntax, see links for the syntax details (default: NULL)

Simple syntax details:

Details: Pull data from Elasticsearch. Everything is done by bulk. This method is essentially a wrapper around '\$search()' with parameter 'head = FALSE'

Returns: a list of datasets corresponding to the pull request, else an error. Keys of the list are index names matching the request, value are the associated tibbles

Examples:

```
\dontrun{
# push some data sample
kc$push(dplyr::storms, "storms")
# get the whole "sw" index
kc$pull("sw")
# get the whole "sw" index with all metadata
kc$pull("sw", keep_metadata = TRUE)
# get only "name" and "status" columns of indices starting with "s"
# columns not found will be ignored
kc$pull("s*", columns = c("name", "status"))
# limit the size of the result to 10
kc$pull("storms", max_size = 10, bulk_size = 10)
# use Elasticsearch query syntax to select and filter on all indices, for all data
# Here, we want to search for all records taht match the conditions:
# field "height" is strictly more than 180 AND field homeworld is "Tatooine" OR "Naboo"
r <- kc$pull("sw", query = "height:>180 && homeworld:(Tatooine || Naboo)")
# it can be used in conjunction with `columns` to select only columns that matter
r <- kc$pull("sw", query = "height:>180 && homeworld:(Tatooine || Naboo)", columns =
  c("name", "hair_color", "homeworld"))
}
```

Method move():

Usage:

```
Kibior$move(
  from_index,
  to_index,
  from_instance = NULL,
  force = FALSE,
  copy = FALSE
)
```

Arguments:

from_index The source index name (default: NULL).

to_index The destination index name (default: NULL).

from_instance If not NULL, the Kibior object of another instance. if NULL (default), this instance will be used. (default: NULL).

force Does the destination index need to be erase? (default: FALSE)

`copy` Does the destination have to be a copy of the source? FALSE (default) will delete source index, TRUE will keep it. (default: FALSE).

Details: Move data from one index to another. It needs to be configured in the ‘config/elasticsearch.yml’ file to actually work.

Returns: the reindex result

Examples:

```
\dontrun{
kc$push(dplyr::starwars, "sw", mode = "recreate")
# move data from an index to another (change name, same instance)
r <- kc$move(from_index = "sw", to_index = "sw_new")
kc$pull("sw_new")
kc$list()
}
```

Method `copy()`:

Usage:

```
Kibior$copy(from_index, to_index, from_instance = NULL, force = FALSE)
```

Arguments:

`from_index` The source index name (default: NULL).

`to_index` The destination index name (default: NULL).

`from_instance` If not NULL, the Kibior object of another instance. if NULL (default), this instance will be used. (default: NULL).

`force` Does the destination index need to be erase? (default: FALSE)

Details: Copy data from one index to another. It needs to be configured in the ‘config/elasticsearch.yml’ file to actually work. This method is a wrapper around ‘\$move(copy = TRUE)’.

Returns: the reindex result

Examples:

```
\dontrun{
# copy data from one index to another (same instance)
r <- kc$copy(from_index = "sw_new", to_index = "sw")
kc$pull(c("sw", "sw_new"))
kc$list()
}
```

Method `match()`:

Usage:

```
Kibior$match(index_name)
```

Arguments:

`index_name` the index name to use in Elasticsearch, can be a pattern with ‘*’.

Details: Match requested index names against Elasticsearch indices list.

Returns: a vector of matching index names, NULL if nothing matches.

Examples:

```
\dontrun{
# search "sw" index name
kc$match("sw")
# search all starting with an "s"
kc$match("s*")
# get all index name, identical to `list()`
kc$match("*")
# search multiple names
kc$match(c("sw", "sw_new", "nope"))
# search multiple names with pattern
kc$match(c("s*", "nope"))
}
```

Method search():

Usage:

```
Kibior$search(
  index_name = "_all",
  keep_metadata = FALSE,
  columns = NULL,
  bulk_size = 500,
  max_size = NULL,
  scroll_timer = "3m",
  head = TRUE,
  query = NULL
)
```

Arguments:

index_name the index name to use in Elasticsearch (default: NULL).

keep_metadata does Elasticsearch needs to sent metadata? Data columns will be prefixed by "_source." (default: FALSE).

columns a vector of columns to select (default: NULL (all columns)).

bulk_size the number of record to send to Elasticsearch in a row (default: 500).

max_size the number of record Elasticsearch will send (default: NULL (all data)).

scroll_timer the time the scroll API will let the request alive to scroll on the result (default: "3m" (3 minutes)).

head a boolean limiting the search result and time (default: TRUE)

query a string formatted to Elasticsearch query syntax, see links for the syntax details (default: NULL)

Details: Search data from Elasticsearch. The goal of this method is to discover quickly what data are interesting, thus 'head = TRUE' by default. If you want to get all data, use 'head = FALSE' or '\$pull()'. Everything is done by bulk.

Returns: a list of datasets corresponding to the pull request, else an error. Keys of the list are index names matching the request, value are the associated tibbles

Examples:

```

\dontrun{
# search "sw" index, head mode on
kc$search("sw")
# search "sw" index with all metadata, head mode on
kc$search("sw", keep_metadata = TRUE)
# get only "name" field of the head of indices starting with "s"
# if an index does not have the "name" field, it will be empty
kc$search("s*", columns = "name")
# limit the size of the result to 50 to the whole index
kc$search("storms", max_size = 50, bulk_size = 50, head = FALSE)
# use Elasticsearch query syntax to select and filter on all indices, for all data
# Here, we want to search for all records taht match the conditions:
# field "height" is strictly more than 180 AND field homeworld is "Tatooine" OR "Naboo"
kc$search("*", query = "height:>180 && homeworld:(Tatooine || Naboo)")
# it can be used in conjunction with `columns` to select only columns that matter
kc$search("*", query = "height:>180 && homeworld:(Tatooine || Naboo)", columns =
  c("name", "hair_color", "homeworld"))
}

```

Method inner_join():

Usage:

```
Kibior$inner_join(...)
```

Arguments:

... see 'join()' params.

Details: Execute a inner join between two datasets using 'dplyr' joins. The datasets can be in-memory (variable name) or the name of an currently stored Elasticsearch index. Joins cannot be done on column of type "list" ("by" argument).

Returns: a tibble

Examples:

```

\dontrun{
# some data for joins examples
kc$push(ggplot2::diamonds, "diamonds")
# prepare join datasets, only big the biggest diamonds are selected (9)
sup_carat <- dplyr::filter(ggplot2::diamonds, carat > 3.5)
r <- kc$push(sup_carat, "diamonds_superior")
# execute a inner_join with one index and one in-memory dataset
kc$inner_join(ggplot2::diamonds, "diamonds_superior")
# execute a inner_join with one index queried, and one in-memory dataset
kc$inner_join(ggplot2::diamonds, "diamonds", right_query
  = "carat:>3.5")
}

```

Method full_join():

Usage:


```
Kibior$full_join(...)
```

Arguments:

... see 'join()' params.

Details: Execute a full join between two datasets using 'dplyr' joins. The datasets can be in-memory (variable name) or the name of an currently stored Elasticsearch index. Joins cannot be done on column of type "list" ("by" argument).

Returns: a tibble

Examples:

```
\dontrun{
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a full_join with one index and one in-memory dataset
kc$full_join(fair_cut, "diamonds_superior")
# execute a full_join with one index queried, and one in-memory dataset
kc$full_join(sup_carat, "diamonds", right_query = "cut:fair")
}
```

Method left_join():

Usage:

```
Kibior$left_join(...)
```

Arguments:

... see 'join()' params.

Details: Execute a left join between two datasets using 'dplyr' joins. The datasets can be in-memory (variable name) or the name of an currently stored Elasticsearch index. Joins cannot be done on column of type "list" ("by" argument).

Returns: a tibble

Examples:

```
\dontrun{
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a left_join with one index and one in-memory dataset
kc$left_join(fair_cut, "diamonds_superior")
# execute a left_join with one index queried, and one in-memory dataset
kc$left_join(sup_carat, "diamonds", right_query
  = "cut:fair")
}
```

Method right_join():

Usage:

```
Kibior$right_join(...)
```

Arguments:

... see `'join()'` params.

Details: Execute a right join between two datasets using `'dplyr'` joins. The datasets can be in-memory (variable name) or the name of an currently stored Elasticsearch index. Joins cannot be done on column of type "list" ("by" argument).

Returns: a tibble

Examples:

```
\dontrun{
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a right_join with one index and one in-memory dataset
kc$right_join(fair_cut, "diamonds_superior")
# execute a right_join with one index queried, and one in-memory dataset
kc$right_join(sup_carat, "diamonds", right_query
  = "cut:fair")
}
```

Method `semi_join()`:*Usage:*

```
Kibior$semi_join(...)
```

Arguments:

... see `'join()'` params.

Details: Execute a semi join between two datasets using `'dplyr'` joins. The datasets can be in-memory (variable name) or the name of an currently stored Elasticsearch index. Joins cannot be done on column of type "list" ("by" argument).

Returns: a tibble

Examples:

```
\dontrun{
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a semi_join with one index and one in-memory dataset
kc$semi_join(fair_cut, "diamonds_superior")
# execute a semi_join with one index queried, and one in-memory dataset
kc$semi_join(sup_carat, "diamonds", right_query
  = "cut:fair")
}
```

Method `anti_join()`:*Usage:*

```
Kibior$anti_join(...)
```

Arguments:

... see 'join()' params.

Details: Execute a anti join between two datasets using 'dplyr' joins. The datasets can be in-memory (variable name) or the name of an currently stored Elasticsearch index. Joins cannot be done on column of type "list" ("by" argument).

Returns: a tibble

Examples:

```
\dontrun{
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a anti_join with one index and one in-memory dataset
kc$anti_join(fair_cut, "diamonds_superior")
# execute a anti_join with one index queried, and one in-memory dataset
kc$anti_join(sup_carat, "diamonds", right_query
  = "cut:fair")
#
# Do not mind this, removing example indices
elastic::index_delete(kc$connection, "*")
kc <- NULL
}
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
Kibior$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

Régis Ongaro-Carcy, <regis.ongaro-carcy2@crchudequebec.ulaval.ca>

References

Kibio.science: <http://kibio.science>,

Elasticsearch documentation: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

See Also

[kibior](#)

you should use [count](#) for more accurate count.

<https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#time-units> for time-units and

<https://www.elastic.co/guide/en/elasticsearch/reference/>

[current/query-dsl-query-string-query.html#query-string-syntax](https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html#query-string-syntax) for the Elasticsearch query string syntax.

: <https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-reindex.html>
Elasticsearch reindex feature for more information.

: <https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-reindex.html>
Elasticsearch reindex feature for more information.

<https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html#time-units> for time-units and <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html#query-string-syntax> for the Elasticsearch query string syntax.

Examples

```
## -----
## Method `Kibior$new`
## -----

## Not run:
# default initialization, connect to "localhost:9200"
kc <- Kibior$new()
# connect to "192.168.2.145:9200"
kc <- Kibior$new("192.168.2.145")
# connect to "es:15005", verbose mode activated
kc <- Kibior$new(host = "elasticsearch", port = 15005, verbose = TRUE)
# connect to "192.168.2.145:9450" with credentials "foo:bar"
kc <- Kibior$new(host = "192.168.2.145", port = 9450, user = "foo", pwd = "bar")
# connect to "elasticsearch:9200"
kc <- Kibior$new("elasticsearch")

# get kibior var from env (".Renviro" file or local env)
dd <- system.file("doc_env", "kibior_build.R", package = "kibior")
source(dd, local = TRUE)
kc <- .kibior_get_instance_from_env()
kc$quiet_progress <- TRUE

# preparing all examples (do not mind this for this method)
delete_if_exists <- function(index_names){
  tryCatch(
    expr = { kc$delete(index_names) },
    error = function(e){ }
  )
}
delete_if_exists(c(
  "aaa",
  "bbb",
  "ccc",
  "ddd",
  "sw",
  "sw_naboo",
  "sw_tatooine",
```

```
        "sw_alderaan",
        "sw_from_file",
        "storms",
        "starwars"
    ))

## End(Not run)

## -----
## Method `Kibior$print`
## -----

## Not run:
print(kc)

## End(Not run)

## -----
## Method `Kibior$eq`
## -----

## Not run:
kc$eq(kc)

## End(Not run)

## -----
## Method `Kibior$ne`
## -----

## Not run:
kc$ne(kc)

## End(Not run)

## -----
## Method `Kibior$create`
## -----

## Not run:
kc$create("aaa")
kc$create(c("bbb", "ccc"))

## End(Not run)

## -----
```

```
## Method `Kibior$list`
## -----

## Not run:
kc$list()
kc$list(get_specials = TRUE)

## End(Not run)

## -----
## Method `Kibior$has`
## -----

## Not run:
kc$has("aaa")
kc$has(c("bbb", "ccc"))

## End(Not run)

## -----
## Method `Kibior$delete`
## -----

## Not run:
kc$delete("aaa")
kc$delete(c("bbb", "ccc"))

## End(Not run)

## -----
## Method `Kibior$add_description`
## -----

## Not run:
kc$add_description(
  index_name = "sw",
  dataset_name = "starwars",
  source_name = "Package dplyr",
  index_description = "Description of starwars characters, the data comes from the Star Wars API.",
  version = "dplyr (1.0.0)",
  link = "http://swapi.dev/",
  direct_download_link = "http://swapi.dev/",
  version_date = "2020-05-28",
  license_link = "MIT",
  columns = list(
    "name" = "Name of the character",
    "height" = "Height (cm)",
    "mass" = "Weight (kg)",
    "hair_color" = "Hair colors",
```

```

        "skin_color" = "Skin colors",
        "eye_color" = "Eye colors",
        "birth_year" = "Year born (BBY = Before Battle of Yavin)",
        "sex" = "The biological sex of the character, namely male, female,
                hermaphroditic, or none (as in the case for Droids).",
        "gender" = "The gender role or gender identity of the character as determined by
                their personality or the way they were programmed (as in the case for Droids
                ).",
        "homeworld" = "Name of homeworld",
        "species" = "Name of species",
        "films" = "List of films the character appeared in",
        "vehicles" = "List of vehicles the character has piloted",
        "starships" = "List of starships the character has piloted"
    )
)

## End(Not run)

## -----
## Method `Kibior$has_description`
## -----

## Not run:
kc$has_description("s*")
kc$has_description(c("sw", "asdf"))

## End(Not run)

## -----
## Method `Kibior$missing_descriptions`
## -----

## Not run:
kc$missing_descriptions()

## End(Not run)

## -----
## Method `Kibior$remove_description`
## -----

## Not run:
# remove the description of 'test' index
kc$remove_description("test")

## End(Not run)

## -----
## Method `Kibior$clean_descriptions`

```

```
## -----  
  
## Not run:  
# remove the description of 'test' index  
kc$clean_descriptions()  
  
## End(Not run)  
  
## -----  
## Method `Kibior$describe`  
## -----  
  
## Not run:  
kc$describe("s*")  
kc$describe("sw", columns = c("name", "height"))  
  
## End(Not run)  
  
## -----  
## Method `Kibior$describe_index`  
## -----  
  
## Not run:  
kc$describe_index("s*")  
  
## End(Not run)  
  
## -----  
## Method `Kibior$describe_columns`  
## -----  
  
## Not run:  
kc$describe_columns("s*", c("name", "height"))  
  
## End(Not run)  
  
## -----  
## Method `Kibior$infos`  
## -----  
  
## Not run:  
kc$infos()  
  
## End(Not run)  
  
## -----  
## Method `Kibior$ping`  
## -----
```



```
## Not run:
kc$ping()

## End(Not run)

## -----
## Method `Kibior$mappings`
## -----

## Not run:
kc$mappings()
kc$mappings("sw")
kc$mappings(c("sw", "sw_naboo"))

## End(Not run)

## -----
## Method `Kibior$settings`
## -----

## Not run:
kc$settings()
kc$settings("sw")
kc$settings(c("sw", "sw_tatooine"))

## End(Not run)

## -----
## Method `Kibior$aliases`
## -----

## Not run:
kc$aliases()
kc$aliases("sw")
kc$aliases(c("sw", "sw_alderaan"))

## End(Not run)

## -----
## Method `Kibior$dim`
## -----

## Not run:
# Couple [<nb obs> <nb var>] in "sw"
kc$dim("sw")
# Couple [<nb obs> <nb var>] in indices "sw_naboo" and "sw_alderaan"
kc$dim(c("sw_naboo", "sw_alderaan"))
```

```

## End(Not run)

## -----
## Method `Kibior$columns`
## -----

## Not run:
kc$columns("sw")          # direct search
kc$columns("sw_*")        # pattern search

## End(Not run)

## -----
## Method `Kibior$count`
## -----

## Not run:
# Number of observations (nb of records) in "sw"
kc$count("sw")
# Number of observations in indices "sw_naboo" and "sw_tatooine"
kc$count(c("sw_naboo", "sw_tatooine"))
# Number of variables (nb of columns) in index "sw_naboo"
kc$count("sw_naboo", type = "variables")

## End(Not run)

## -----
## Method `Kibior$avg`
## -----

## Not run:
# Avg of "sw" column "height"
kc$avg("sw", "height")
# if pattern
kc$avg("s*", "height")
# multiple indices, multiple columns
kc$avg(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$mean`
## -----

## Not run:
# mean of "sw" column "height"
kc$mean("sw", "height")
# if pattern
kc$mean("s*", "height")

```

```

# multiple indices, multiple columns
kc$mean(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$min`
## -----

## Not run:
# min of "sw" column "height"
kc$min("sw", "height")
# if pattern
kc$min("s*", "height")
# multiple indices, multiple columns
kc$min(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$max`
## -----

## Not run:
# max of "sw" column "height"
kc$max("sw", "height")
# if pattern
kc$max("s*", "height")
# multiple indices, multiple columns
kc$max(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$sum`
## -----

## Not run:
# sum of "sw" column "height"
kc$sum("sw", "height")
# if pattern
kc$sum("s*", "height")
# multiple indices, multiple columns
kc$sum(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$stats`

```

```

## -----

## Not run:
# Stats of "sw" column "height"
kc$stats("sw", "height")
# if pattern
kc$stats("s*", "height")
# multiple indices and sigma definition
kc$stats(c("sw", "sw2"), "height", sigma = 2.5)
# multiple indices, multiple columns
kc$stats(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$percentiles`
## -----

## Not run:
# percentiles of "sw" column "height", default is with q1, q2 and q3
kc$percentiles("sw", "height")
# if pattern
kc$percentiles("s*", "height")
# defining percents to get
kc$percentiles("s*", "height", percents = c(20, 25))
# multiple indices, multiple columns
kc$percentiles(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$q1`
## -----

## Not run:
# Q1 of "sw" column "height"
kc$q1("sw", "height")
# if pattern
kc$q1("s*", "height")
# multiple indices, multiple columns
kc$q1(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$q2`
## -----

## Not run:
# Q2 of "sw" column "height"

```

```

kc$q2("sw", "height")
# if pattern
kc$q2("s*", "height")
# multiple indices, multiple columns
kc$q2(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$median`
## -----

## Not run:
# median of "sw" column "height"
kc$median("sw", "height")
# if pattern
kc$median("s*", "height")
# multiple indices, multiple columns
kc$median(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$q3`
## -----

## Not run:
# Q3 of "sw" column "height"
kc$q3("sw", "height")
# if pattern
kc$q3("s*", "height")
# multiple indices, multiple columns
kc$q3(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

## -----
## Method `Kibior$summary`
## -----

## Not run:
# summary of "sw" column "height"
kc$summary("sw", "height")
# if pattern
kc$summary("s*", "height")
# multiple indices, multiple columns
kc$summary(c("sw", "sw2"), c("height", "mass"), query = "homeworld:naboo")

## End(Not run)

```

```
## -----  
## Method `Kibior$keys`  
## -----  
  
## Not run:  
kc$keys("sw", "name")  
kc$keys("sw", "eye_color")  
  
## End(Not run)  
  
## -----  
## Method `Kibior$bam_to_tibble`  
## -----  
  
## Not run:  
dd_bai <- system.file("extdata", "test.bam.bai", package = "kibior")  
bam_param <- Rsamtools::ScanBamParam(what = c("pos", "qwidth"))  
bam_data <- Rsamtools::scanBam(dd_bai, param = bam_param)  
kc$bam_to_tibble(bam_data)  
  
## End(Not run)  
  
## -----  
## Method `Kibior$soft_cast`  
## -----  
  
## Not run:  
kc$soft_cast(datasets::iris)  
  
## End(Not run)  
  
## -----  
## Method `Kibior$get_resource`  
## -----  
  
## Not run:  
kc$get_resource(system.file("R", "kibior.R", package = "kibior"))  
kc$get_resource("https://ftp.ncbi.nlm.nih.gov/entrez/README")  
  
## End(Not run)  
  
## -----  
## Method `Kibior$export`  
## -----  
  
## Not run:  
f <- tempfile(fileext=".csv")  
# export and overwrite last file with the same data from Elasticsearch
```

```
kc$export(data = "sw", filepath = f)
# export from in-memory data to a file
kc$export(data = dplyr::starwars, filepath = f, force = TRUE)

## End(Not run)

## -----
## Method `Kibior$import_tabular`
## -----

## Not run:
f <- tempfile(fileext = ".csv")
rio::export(ggplot2::diamonds, f)
# import to in-memory variable
kc$import_tabular(filepath = f)
# import raw data
kc$import_tabular(filepath = f, to_tibble = FALSE)

## End(Not run)

## -----
## Method `Kibior$import_features`
## -----

## Not run:
# get sample files
f_gff <- system.file("extdata", "chr_y.gff3.gz", package = "kibior")
f_bed <- system.file("extdata", "cpg.bed", package = "kibior")
# import to in-memory variable
kc$import_features(filepath = f_bed)
kc$import_features(filepath = f_gff)
# import raw data
kc$import_features(filepath = f_bed, to_tibble = FALSE)
kc$import_features(filepath = f_gff, to_tibble = FALSE)

## End(Not run)

## -----
## Method `Kibior$import_alignments`
## -----

## Not run:
# get sample file
f_bai <- system.file("extdata", "test.bam.bai", package = "kibior")
# import to in-memory variable
kc$import_alignments(filepath = f_bai)
# import raw data
kc$import_alignments(filepath = f_bai, to_tibble = FALSE)

## End(Not run)
```

```

## -----
## Method `Kibior$import_json`
## -----

## Not run:
# get sample file
f_json <- system.file("extdata", "storms100.json", package = "kibior")
# import to in-memory variable
kc$import_json(f_json)
# import raw data
kc$import_json(f_json, to_tibble = FALSE)

## End(Not run)

## -----
## Method `Kibior$import_sequences`
## -----

## Not run:
# get sample file
f_dna <- system.file("extdata", "dna_human_y.fa.gz", package = "kibior")
f_rna <- system.file("extdata", "ncrna_mus_musculus.fa.gz", package = "kibior")
f_aa <- system.file("extdata", "pep_mus_spretus.fa.gz", package = "kibior")
# import to in-memory variable
kc$import_sequences(filepath = f_dna, fasta_type = "dna")
# import raw data
kc$import_sequences(filepath = f_rna, to_tibble = FALSE, fasta_type = "rna")
# import auto
kc$import_sequences(filepath = f_aa)

## End(Not run)

## -----
## Method `Kibior$guess_import`
## -----

## Not run:
# get sample file
f_dna <- system.file("extdata", "dna_human_y.fa.gz", package = "kibior")
f_rna <- system.file("extdata", "ncrna_mus_musculus.fa.gz", package = "kibior")
f_aa <- system.file("extdata", "pep_mus_spretus.fa.gz", package = "kibior")
f_bai <- system.file("extdata", "test.bam.bai", package = "kibior")
f_gff <- system.file("extdata", "chr_y.gff3.gz", package = "kibior")
f_bed <- system.file("extdata", "cpg.bed", package = "kibior")
# import
kc$guess_import(f_dna)
kc$guess_import(f_rna)
kc$guess_import(f_aa)
kc$guess_import(f_bai)

```



```

kc$guess_import(f_gff)
kc$guess_import(f_bed)

## End(Not run)

## -----
## Method `Kibior$import`
## -----

## Not run:
# get sample file
f_aa <- system.file("extdata", "pep_mus_spretus.fa.gz", package = "kibior")
f_gff <- system.file("extdata", "chr_y.gff3.gz", package = "kibior")
f_bai <- system.file("extdata", "test.bam.bai", package = "kibior")
# import
kc$import(filepath = f_aa)
# import to Elasticsearch index ("sw_from_file") if not exists
kc$import(filepath = f_bai, push_index = "sw_from_file")
# import to index by recreating it, then pull indexed data
kc$import(filepath = f_gff, push_index = "sw_from_file",
  push_mode = "recreate")

## End(Not run)

## -----
## Method `Kibior$push`
## -----

## Not run:
# erase the last push data by recreating the index and re-pushing data
kc$push(dplyr::starwars, index_name = "sw", mode = "recreate")
# characters names are unique, can be used as ID
kc$push(dplyr::starwars, index_name = "sw", mode = "recreate", id_col = "name")
# a bit more complicated: update some data of the dataset "starwars"
# 38 records on 87 filtered
some_new_data <- dplyr::filter(dplyr::starwars, height > 180)
# make them all "gender <- female"
some_new_data["gender"] <- "female"
# update that apply, based on cahracter names to match the right record
kc$push(some_new_data, "sw", mode = "update", id_col = "name")
# view result by querying
kc$pull("sw", query = "height:>180", columns = c("name", "gender"))

## End(Not run)

## -----
## Method `Kibior$pull`
## -----

## Not run:

```

```

# push some data sample
kc$push(dplyr::storms, "storms")
# get the whole "sw" index
kc$pull("sw")
# get the whole "sw" index with all metadata
kc$pull("sw", keep_metadata = TRUE)
# get only "name" and "status" columns of indices starting with "s"
# columns not found will be ignored
kc$pull("s*", columns = c("name", "status"))
# limit the size of the result to 10
kc$pull("storms", max_size = 10, bulk_size = 10)
# use Elasticsearch query syntax to select and filter on all indices, for all data
# Here, we want to search for all records taht match the conditions:
# field "height" is strictly more than 180 AND field homeworld is "Tatooine" OR "Naboo"
r <- kc$pull("sw", query = "height:>180 && homeworld:(Tatooine || Naboo)")
# it can be used in conjunction with `columns` to select only columns that matter
r <- kc$pull("sw", query = "height:>180 && homeworld:(Tatooine || Naboo)", columns =
  c("name", "hair_color", "homeworld"))

## End(Not run)

## -----
## Method `Kibior$move`
## -----

## Not run:
kc$push(dplyr::starwars, "sw", mode = "recreate")
# move data from an index to another (change name, same instance)
r <- kc$move(from_index = "sw", to_index = "sw_new")
kc$pull("sw_new")
kc$list()

## End(Not run)

## -----
## Method `Kibior$copy`
## -----

## Not run:
# copy data from one index to another (same instance)
r <- kc$copy(from_index = "sw_new", to_index = "sw")
kc$pull(c("sw", "sw_new"))
kc$list()

## End(Not run)

## -----
## Method `Kibior$match`
## -----

```

```

## Not run:
# search "sw" index name
kc$match("sw")
# search all starting with an "s"
kc$match("s*")
# get all index name, identical to `$list()`
kc$match("*")
# search multiple names
kc$match(c("sw", "sw_new", "nope"))
# search multiple names with pattern
kc$match(c("s*", "nope"))

## End(Not run)

## -----
## Method `Kibior$search`
## -----

## Not run:
# search "sw" index, head mode on
kc$search("sw")
# search "sw" index with all metadata, head mode on
kc$search("sw", keep_metadata = TRUE)
# get only "name" field of the head of indices starting with "s"
# if an index does not have the "name" field, it will be empty
kc$search("s*", columns = "name")
# limit the size of the result to 50 to the whole index
kc$search("storms", max_size = 50, bulk_size = 50, head = FALSE)
# use Elasticsearch query syntax to select and filter on all indices, for all data
# Here, we want to search for all records taht match the conditions:
# field "height" is strictly more than 180 AND field homeworld is "Tatooine" OR "Naboo"
kc$search("*", query = "height:>180 && homeworld:(Tatooine || Naboo)")
# it can be used in conjunction with `columns` to select only columns that matter
kc$search("*", query = "height:>180 && homeworld:(Tatooine || Naboo)", columns =
  c("name", "hair_color", "homeworld"))

## End(Not run)

## -----
## Method `Kibior$inner_join`
## -----

## Not run:
# some data for joins examples
kc$push(ggplot2::diamonds, "diamonds")
# prepare join datasets, only big the biggest diamonds are selected (9)
sup_carat <- dplyr::filter(ggplot2::diamonds, carat > 3.5)
r <- kc$push(sup_carat, "diamonds_superior")
# execute a inner_join with one index and one in-memory dataset
kc$inner_join(ggplot2::diamonds, "diamonds_superior")
# execute a inner_join with one index queried, and one in-memory dataset

```

```

kc$inner_join(ggplot2::diamonds, "diamonds", right_query
  = "carat:>3.5")

## End(Not run)

## -----
## Method `Kibior$full_join`
## -----

## Not run:
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a full_join with one index and one in-memory dataset
kc$full_join(fair_cut, "diamonds_superior")
# execute a full_join with one index queried, and one in-memory dataset
kc$full_join(sup_carat, "diamonds", right_query = "cut:fair")

## End(Not run)

## -----
## Method `Kibior$left_join`
## -----

## Not run:
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a left_join with one index and one in-memory dataset
kc$left_join(fair_cut, "diamonds_superior")
# execute a left_join with one index queried, and one in-memory dataset
kc$left_join(sup_carat, "diamonds", right_query
  = "cut:fair")

## End(Not run)

## -----
## Method `Kibior$right_join`
## -----

## Not run:
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a right_join with one index and one in-memory dataset
kc$right_join(fair_cut, "diamonds_superior")
# execute a right_join with one index queried, and one in-memory dataset
kc$right_join(sup_carat, "diamonds", right_query
  = "cut:fair")

```

```

## End(Not run)

## -----
## Method `Kibior$semi_join`
## -----

## Not run:
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a semi_join with one index and one in-memory dataset
kc$semi_join(fair_cut, "diamonds_superior")
# execute a semi_join with one index queried, and one in-memory dataset
kc$semi_join(sup_carat, "diamonds", right_query
  = "cut:fair")

## End(Not run)

## -----
## Method `Kibior$anti_join`
## -----

## Not run:
# prepare join datasets, fair cuts
fair_cut <- dplyr::filter(ggplot2::diamonds, cut == "Fair") # 1605 lines
sup_carat <- kc$pull("diamonds_superior")$diamonds_superior
# execute a anti_join with one index and one in-memory dataset
kc$anti_join(fair_cut, "diamonds_superior")
# execute a anti_join with one index queried, and one in-memory dataset
kc$anti_join(sup_carat, "diamonds", right_query
  = "cut:fair")
#
# Do not mind this, removing example indices
elastic::index_delete(kc$connection, "*")
kc <- NULL

## End(Not run)

```

Kibior equals operator

Kibior equals operator

Description

Kibior equals operator

Usage

```
## S3 method for class 'KibiorOperators'  
x == y
```

Arguments

x	the first Kibior instance
y	the second Kibior instance

Details

Call `kibiorR '$eq()'` for a comparison of the two instances.

Value

TRUE if the two instances are equals, else FALSE

See Also

Other comparison: [Kibior not-equals operator](#), [Static - Kibior is instance](#)

Kibior not-equals operator

Kibior not-equals operator

Description

Kibior not-equals operator

Usage

```
## S3 method for class 'KibiorOperators'  
x != y
```

Arguments

x	the first Kibior instance
y	the second Kibior instance

Details

Call `kibiorR '$ne()'` for a comparison of the two instances.

Value

TRUE if the two instances are different, else FALSE

See Also

Other comparison: [Kibior equals operator](#), [Static - Kibior is instance](#)

Static - initiate a direct instance to Kibio public repository
Static - initiate a direct instance to Kibio public repository

Description

Static - initiate a direct instance to Kibio public repository

Arguments

verbose verbosity activation (default: FALSE)

Details

Initiate a instance of Kibior connected to the Kibio public repository.

Value

a new instance of Kibior conencted to Kibio service

Static - Kibior is instance
Static - Kibior is instance

Description

Static - Kibior is instance

Arguments

obj an object

Details

Tests if a given object is a Kibior instance. Basically compute symmetric difference between two sets of class.

Value

TRUE if the given object is an instance of Kibior, else FALSE

See Also

Other comparison: [Kibior equals operator](#), [Kibior not-equals operator](#)

Static - Tests if packages are installed
Static - Tests if packages are installed

Description

Static - Tests if packages are installed

Arguments

pkg_names a vector of some package names

Details

Get the installation status of some package names (installed TRUE/FALSE).

Value

a named vector of packages with installation status

Index

- !=.KibiorOperators (Kibior not-equals operator), [54](#)
- * **cluster-wealth**
 - kibior, [2](#)
- * **comparison**
 - Kibior equals operator, [53](#)
 - Kibior not-equals operator, [54](#)
 - Static - Kibior is instance, [55](#)
- * **crud-index**
 - kibior, [2](#)
- * **crud-metadata**
 - kibior, [2](#)
- * **data-manipulation**
 - kibior, [2](#)
- * **dataset**
 - kibior, [2](#)
- * **data**
 - kibior, [2](#)
- * **initiate**
 - Static - initiate a direct instance to Kibio public repository, [55](#)
- * **install**
 - Static - Tests if packages are installed, [56](#)
- * **integration**
 - kibior, [2](#)
- * **joins**
 - kibior, [2](#)
- * **kibior-metadata**
 - kibior, [2](#)
- * **move-data**
 - kibior, [2](#)
- * **search-data**
 - kibior, [2](#)
- * **stats**
 - kibior, [2](#)
- ==.KibiorOperators (Kibior equals operator), [53](#)
- count, [35](#)
- Kibior (kibior), [2](#)
- kibior, [2](#), [35](#)
- Kibior equals operator, [53](#)
- Kibior not-equals operator, [54](#)
- R6Class, [2](#)
- Static - initiate a direct instance to Kibio public repository, [55](#)
- Static - Kibior is instance, [55](#)
- Static - Tests if packages are installed, [56](#)