Title Feature Stores for the 'diseasy' Framework
Version 0.1
Description Simple feature stores and tools for creating personalised feature stores.
'diseasystore' powers feature stores which can automatically link and aggregate features to a given stratification level. These feature stores are automatically time-versioned (powered by the 'SCDB' package) and allows you to easily and dynamically compute features as part of your continuous integration.

License GPL (>= 3)
Encoding UTF-8
RoxygenNote 7.2.3
Language en-GB
Imports checkmate, DBI, dbplyr, dplyr, glue, lintr, lubridate, purrr, readr, rlang, R6, SCDB, stringr, tidy, tidyselect, zoo
Suggests curl, knitr, R.utils, rmarkdown, RSQLite, RPostgres, testthat (>= 3.0.0), tibble, spelling, withr
VignetteBuilder knitr
URL https://github.com/ssi-dk/diseasystore,
https://ssi-dk.github.io/diseasystore/
BugReports https://github.com/ssi-dk/diseasystore/issues
Config/testthat/edition 3
NeedsCompilation no
Author Rasmus Skytte Randløv [aut, cre]
(Marcus Munch Grünwald [ctb] (https://orcid.org/0000-0006-8090-406X),
Kaare Grønbøll [ctb] (https://orcid.org/0000-0002-6258-8212),
Kasper Schou Telkamp [ctb] (https://orcid.org/0009-0001-5126-0190),
Lasse Engbo Christiansen [ctb]
(https://orcid.org/0000-0001-5019-1931),
Statens Serum Institut, SSI [cph, fnd]
**Description**

Provides age_labels that follows the mg standard

**Usage**

```r
age_labels(age_cuts)
```

**Arguments**

- `age_cuts`: The lower bound of the groups (0 is implicitly included)

**Value**

A vector of labels with zero-padded numerics so they can be sorted easily

**Examples**

```r
age_labels(c(5, 12, 20, 30))
```
### aggregators

#### Feature aggregators

<table>
<thead>
<tr>
<th>Description</th>
<th>Feature aggregators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>key_join_sum(.data, feature)</td>
</tr>
<tr>
<td></td>
<td>key_join_max(.data, feature)</td>
</tr>
<tr>
<td></td>
<td>key_join_min(.data, feature)</td>
</tr>
<tr>
<td></td>
<td>key_join_count(.data, feature)</td>
</tr>
</tbody>
</table>

#### Arguments

<table>
<thead>
<tr>
<th>.data</th>
<th>(any)</th>
<th>The data object to perform the operation on.</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature</td>
<td>(character)</td>
<td>Name of the feature to perform the aggregation over</td>
</tr>
</tbody>
</table>

#### Value

A `dplyr::summarise` to aggregate the features together using the given function (sum/max/min/count)

#### Examples

# Primarily used within the framework but can be used individually:

data <- dplyr::mutate(mtcars, key_name = rownames(mtcars), .before = dplyr::everything())

key_join_sum(data, "mpg")  # sum(mtcars$mpg)
key_join_max(data, "mpg")  # max(mtcars$mpg)
key_join_min(data, "mpg")  # min(mtcars$mpg)
key_join_count(data, "mpg")  # nrow(mtcars)
available_diseasystores

Detect available diseasystores

Description
Detect available diseasystores

Usage
available_diseasystores()

Value
The installed diseasystores on the search path

Examples
available_diseasystores()  # DiseasystoreGoogleCovid19 + more from other packages

diseasyoption

Helper function to get option

Description
Helper function to get option

Usage
diseasyoption(option, class = "DiseasystoreBase")

Arguments

- option (character)
  Name of the option to get

- class (character or R6::R6class Diseasy* instance)
  Either the classname or the object the option applies to.

Value
The most specific option within the diseasy framework for the given option and class
Examples

# Retrieve default option for source conn
diseasyoption("source_conn")

# Retrieve DiseasystoreGoogleCovid19 specific option for source conn
diseasyoption("source_conn", "DiseasystoreGoogleCovid19")

# Try to retrieve specific option for source conn for a non existent / un-configured diseasystore
diseasyoption("source_conn", "DiseasystoreNonExistent") # Returns default source_conn

__DiseasystoreBase__

_diseasystore base handler_

Description

This __DiseasystoreBase__ R6 class forms the basis of all feature stores. It defines the primary methods of each feature stores as well as all of the public methods.

Value

A new instance of the __DiseasystoreBase__ R6 class.

Active bindings

- `ds_map` (named list(character))
  A list that maps features known by the feature store to the corresponding feature handlers that compute the features. Read only.
- `available_features` (character)
  A list of available features in the feature store. Read only.
- `label` (character)
  A human readable label of the feature store. Read only.
- `source_conn` (DBIConnection or file path)
  Used to specify where data is located. Read only. Can be DBIConnection or file path depending on the diseasystore.
- `target_conn` (DBIConnection)
  A database connection to store the computed features in. Read only.
- `target_schema` (character)
  The schema to place the feature store in. Read only. If the database backend does not support schema, the tables will be prefixed with target_schema.
- `start_date` (Date)
  Study period start. Read only.
- `end_date` (Date)
  Study period end. Read only.
- `slice_ts` (Date or character)
  Date to slice the database on (used if source_conn is a database). Read only.
Methods

Public methods:

- `DiseasystoreBase$new()`
- `DiseasystoreBase$finalize()`
- `DiseasystoreBase$get_feature()`
- `DiseasystoreBase$key_join_features()`
- `DiseasystoreBase$clone()`

Method `new()`: Creates a new instance of the `DiseasystoreBase` R6 class.

**Usage:**

```r
DiseasystoreBase$new(
  start_date = NULL,
  end_date = NULL,
  slice_ts = NULL,
  source_conn = NULL,
  target_conn = NULL,
  target_schema = NULL,
  verbose = diseasyoption("verbose", self)
)
```

**Arguments:**

- `start_date` (Date)
  - Study period start.
- `end_date` (Date)
  - Study period end.
- `slice_ts` (Date or character)
  - Date to slice the database on (used if `source_conn` is a database).
- `source_conn` (DBIConnection or file path)
  - Used to specify where data is located. Can be DBIConnection or file path depending on the `diseasystore`.
- `target_conn` (DBIConnection)
  - A database connection to store the computed features in.
- `target_schema` (character)
  - The schema to place the feature store in. If the database backend does not support schema, the tables will be prefixed with `target_schema`.
- `verbose` (boolean)
  - Boolean that controls enables debugging information.

**Returns:** A new instance of the `DiseasystoreBase` R6 class.

Method `finalize()`: Closes the open DB connection when removing the object

**Usage:**

```r
DiseasystoreBase$finalize()
```

Method `get_feature()`: Computes, stores, and returns the requested feature for the study period.

**Usage:**

```r
DiseasystoreBase$get_feature()
```
DiseasystoreBase

DiseasystoreBase$get_feature(
    feature,
    start_date = self %.% start_date,
    end_date = self %.% end_date,
    slice_ts = self %.% slice_ts
)

Arguments:
feature (character)
The name of a feature defined in the feature store.
start_date (Date)
Study period start.
end_date (Date)
Study period end.
slice_ts (Date or character)
Date to slice the database on (used if source_conn is a database).

Returns: A tbl_dbi with the requested feature for the study period.

Method key_join_features(): Joins various features from feature store assuming a primary feature (observable) that contains keys to which the secondary features (defined by stratification) can be joined.

Usage:
DiseasystoreBase$key_join_features(
    observable,
    stratification,
    start_date = self %.% start_date,
    end_date = self %.% end_date
)

Arguments:
observable (character)
The name of a feature defined in the feature store
stratification (list(quosures))
Expressions in stratification are evaluated to find appropriate features. These are then joined to the observable feature before stratification is performed.
start_date (Date)
Study period start.
end_date (Date)
Study period end.

Returns: A tbl_dbi with the requested joined features for the study period.

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

Usage:
DiseasystoreBase$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
## DiseasystoreGoogleCovid19

### Description

This `DiseasystoreGoogleCovid19` R6 brings support for using the Google Health COVID-19 Open Data repository. See the vignette("google_covid_19_data") for details on how to configure the feature store.

### Value


### Super class

`diseasystore::DiseasystoreBase` -> `DiseasystoreGoogleCovid19`

### Methods

#### Public methods:

- `DiseasystoreGoogleCovid19$clone()`

#### Method `clone()`:

The objects of this class are cloneable with this method.

**Usage:**

```
DiseasystoreGoogleCovid19$clone(deep = FALSE)
```

**Arguments:**

- `deep` : Whether to make a deep clone.

#### Examples

```r
ds <- DiseasystoreGoogleCovid19$new(source_conn = ".",
                               target_conn = DBI::dbConnect(RSQLite::SQLite()))
rm(ds)
```
**diseasystore_exists**  
*Check for the existence of a diseasystore for the case definition*

**Description**

Check for the existence of a diseasystore for the case definition

**Usage**

```r
diseasystore_exists(label)
```

**Arguments**

- **label**  
  (character)  
  A character string that controls which feature store to get data from.

**Value**

TRUE if the given diseasystore can be matched to a diseasystore on the search path. FALSE otherwise.

**Examples**

```r
diseasystore_exists("Google COVID-19")  # TRUE
diseasystore_exists("Non existent diseasystore")  # FALSE
```

---

**diseasy_linters**  
*The custom linters of diseasy*

**Description**

- **nolint_position_linter**: Check that the `nolint:` statements occur after the character limit  
- **nolint_line_length_linter**: Check that lines adhere to a given character limit, ignoring `nolint` statements

**Usage**

```r
nolint_position_linter(length = 80L)
nolint_line_length_linter(length = 80L)
```

**Arguments**

- **length**  
  maximum line length allowed. Default is 80L (Hollerith limit).
Value

A list of `lint::Lint`

See Also

- `lint::linters` for a complete list of linters available in lintr.
- `https://style.tidyverse.org/syntax.html#long-lines`

Examples

```r
## nolint_position_linter
# will produce lints
lintr::lint(
  text = paste0(strrep("x", 15L), "# nolint: object_name_linter"),
  linters = c(nolint_position_linter(length = 20L), lintr::object_name_linter())
)

# okay
lintr::lint(
  text = paste0(strrep("x", 20L), "# nolint: object_name_linter"),
  linters = c(nolint_position_linter(length = 20L), lintr::object_name_linter())
)

## nolint_line_length_linter
# will produce lints
lintr::lint(
  text = paste0(strrep("x", 25L), "# nolint: object_name_linter."),
  linters = c(nolint_line_length_linter(length = 20L), lintr::object_name_linter())
)

# okay
lintr::lint(
  text = paste0(strrep("x", 20L), "# nolint: object_name_linter."),
  linters = c(nolint_line_length_linter(length = 20L), lintr::object_name_linter())
)
```

drop_diseasystore

*Drop feature stores from DB*

Description

Drop feature stores from DB

Usage

```r
drop_diseasystore(
  pattern = NULL,
  schema = diseasyoption("target_schema"),
)```
conn = SCDB::get_connection()
)

**Arguments**

- **pattern**: Pattern to match the tables by
- **schema**: Schema the diseasestore uses to store data in
- **conn**: DB connection

**Value**

No return value, called for side effects

**Examples**

```r
conn <- SCDB::get_connection(drv = RSQLite::SQLite())
drop_diseasestore(conn = conn)
DBI::dbDisconnect(conn)
```

---

### FeatureHandler

**Description**

This FeatureHandler R6 handles individual features for the feature stores. They define the three methods associated with features (compute, get and key_join).

**Value**

A new instance of the FeatureHandler R6 class.

**Active bindings**

- **compute (function)**
  
  A function of the form "function(start_date, end_date, slice_ts, source_conn)". This function should compute the feature from the source connection.

- **get (function)**
  
  A function of the form "function(target_table, slice_ts, target_conn)". This function should retrieve the computed feature from the target connection.

- **key_join (function)**
  
  One of the aggregators from aggregators.
Methods

Public methods:

- FeatureHandler$new()
- FeatureHandler$clone()

Method new(): Creates a new instance of the FeatureHandler R6 class.

Usage:
FeatureHandler$new(compute = NULL, get = NULL, key_join = NULL)

Arguments:

compute (function)
  A function of the form "function(start_date, end_date, slice_ts, source_conn)"). This function should return a data.frame with the computed feature (computed from the source connection). The data.frame should contain the following columns:
  - key_*: One (or more) columns containing keys to link this feature with other features
  - *: One (or more) columns containing the features that are computed
  - valid_from, valid_until: A set of columns containing the time period for which this feature information is valid.

get (function)
  (Optional). A function of the form "function(target_table, slice_ts, target_conn)"). This function should retrieve the computed feature from the target connection.

key_join (function)
  A function like one of the aggregators from aggregators().
  The function should return an expression on the form: dplyr::summarise(.data, dplyr::across(.cols = tidyselect::all_of(feature), .fns = list(n = ~ aggregation function), .names = "{.fn}")
  .groups = "drop")

Returns: A new instance of the FeatureHandler R6 class.

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

Usage:
FeatureHandler$clone(deep = FALSE)

Arguments:

deep  Whether to make a deep clone.

Examples

# The FeatureHandler is typically configured as part of making a new Diseasystore.
# Most often, we need only specify 'compute` and `key_join` to get a functioning FeatureHandler

# In this example we use mtcars as the basis for our features
conn <- SCDB::get_connection(drv = RSQLite::SQLite())

# We use mtcars as our basis. First we add the rownames as an actual column
data <- dplyr::mutate(mtcars, key_name = rownames(mtcars), .before = dplyr::everything())
# Then we add some imaginary times where these cars were produced
data <- dplyr::mutate(data,
  production_start = as.Date(Sys.Date()) + floor(runif(nrow(mtcars)) * 100),
  production_end = production_start + floor(runif(nrow(mtcars)) * 365))

dplyr::copy_to(conn, data, "mtcars")

# In this example, the feature we want is the "maximum miles per gallon"
# The feature in question in the mtcars data set is then "mpg" and when we need to reduce
# our data set, we want to use the "max()" function.

# We first write a compute function for the mpg in our modified mtcars data set
# Our goal is to get the mpg of all cars that were in production at the between start/end_date
compute_mpg <- function(start_date, end_date, slice_ts, source_conn) {
  out <- SCDB::get_table(source_conn, "mtcars", slice_ts = slice_ts) |> 
    dplyr::filter({{ start_date }} <= .data$production_end, 
    .data$production_start <= {{ end_date }}) |> 
    dplyr::transmute("key_name", "mpg", 
    "valid_from" = "production_start", 
    "valid_until" = "production_end")
  return(out)
}

# We can now combine into our FeatureHandler
fh_max_mpg <- FeatureHandler$new(compute = compute_mpg, key_join = key_join_max)

DBI::dbDisconnect(conn)

---

get_diseasystore

Get the diseasystore for the case definition

**Description**

Get the diseasystore for the case definition

**Usage**

get_diseasystore(label)

**Arguments**

- **label** (%character%)

  A character string that controls which feature store to get data from.

**Value**

The diseasystore generator for the diseasystore matching the given label
Examples

d <- get_diseasystore("Google COVID-19") # Returns the DiseasystoreGoogleCovid19 generator

to_diseasystore_case  Transform case definition to PascalCase

Description

Transform case definition to PascalCase

Usage

to_diseasystore_case(label)

Arguments

label (character)
A character string that controls which feature store to get data from.

Value

The given label formatted to match a Diseasystore

Examples

to_diseasystore_case("Google COVID-19")  # DiseasystoreGoogleCovid19

Existence aware pick operator

Description

Existence aware pick operator

Usage

env %.% field

Arguments

env (object)
The object or environment to attempt to pick from

field (character)
The name of the field to pick from env
Value

Error if the field does not exist in env, otherwise it returns field

Examples

```r
# Create a list
t <- list(a = 1, b = 2)

t$a # 1

t $.% a # 1

t$c # NULL

try(t $.% c) # Gives error since "c" does not exist in "t"
```
Index

\%
aggregators, 3, 11
aggregators(), 12
available_diseasystores, 4
diseasy_linters, 9
diseasyoption, 4
diseasystore::DiseasystoreBase, 8
diseasystore_exists, 9
DiseasystoreBase, 5
DiseasystoreGoogleCovid19, 8
drop_diseasystore, 10

FeatureHandler, 11

get_diseasystore, 13

key_join_count (aggregators), 3
key_join_max (aggregators), 3
key_join_min (aggregators), 3
key_join_sum (aggregators), 3

lintr::linters, 10

nolint_line_length_linter
  (diseasy_linters), 9
nolint_position_linter
  (diseasy_linters), 9

R6, 5, 6, 8, 11, 12

to_diseasystore_case, 14