Package ‘covidregionaldata’

July 5, 2021

**Title**  Subnational Data for COVID-19 Epidemiology

**Version**  0.9.2

**Description** An interface to subnational and national level COVID-19 data sourced from both official sources, such as Public Health England in the UK, and from other COVID-19 data collections, including the World Health Organisation (WHO), European Centre for Disease Prevention and Control (ECDC), John Hopkins University (JHU), Google Open Data and others. Designed to streamline COVID-19 data extraction, cleaning, and processing from a range of data sources in an open and transparent way. This allows users to inspect and scrutinise the data, and tools used to process it, at every step. For all countries supported, data includes a daily time-series of cases. Wherever available data is also provided for deaths, hospitalisations, and tests. National level data are also supported using a range of sources as well as line list data and links to intervention data sets.

**License** MIT + file LICENSE

**URL** [https://epiforecasts.io/covidregionaldata/](https://epiforecasts.io/covidregionaldata/), [https://github.com/epiforecasts/covidregionaldata/](https://github.com/epiforecasts/covidregionaldata/)

**BugReports** [https://github.com/epiforecasts/covidregionaldata/issues/](https://github.com/epiforecasts/covidregionaldata/issues/)

**Depends** R (>= 3.5.0)

**Imports** countrycode (>= 1.2.0), dplyr, httr, lifecycle, lubridate, magrittr, memoise, purrr, R6, readxl, rlang, stringr, tibble, tidyr (>= 1.0.0), tidyselect, vroom, withr, xml2

**Suggests** ggplot2, ggspatial, knitr, mockery, pkgdown, rmarkdown, rvest, rworldmap, sf, spelling, testthat (>= 3.0.0), usethis

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**Language** en-gb

**LazyData** true

**RoxygenNote** 7.1.1
NeedsCompilation: no

Author: Joseph Palmer [aut] (https://orcid.org/0000-0002-5593-9352),
        Katharine Sherratt [aut] (https://orcid.org/0000-0003-2049-3423),
        Jonnie Bevan [aut],
        Hamish Gibbs [aut] (https://orcid.org/0000-0003-4413-453X),
        Sophie Meakin [ctb],
        Joel Hellewell [ctb] (https://orcid.org/0000-0003-2683-0849),
        Patrick Barks [ctb],
        Paul Campbell [ctb],
        Flavio Finger [ctb] (https://orcid.org/0000-0002-8613-5170),
        Richard Boyes [ctb] (https://github.com/rboyes),
        Hugo Gruson [ctb] (https://orcid.org/0000-0002-4094-1476),
        Sebastian Funk [aut],
        Sam Abbott [aut, cre] (https://orcid.org/0000-0001-8057-8037)

Maintainer: Sam Abbott <sam.abbott@lshtm.ac.uk>

Repository: CRAN

Date/Publication: 2021-07-05 11:20:06 UTC

R topics documented:

- add_extra_na_cols ........................................... 4
- all_country_data ........................................... 4
- Belgium .................................................. 5
- Brazil .................................................... 6
- calculate_columns_from_existing_data ..................... 8
- Canada .................................................... 9
- check_level ............................................. 10
- Colombia ............................................... 10
- colombia_codes ......................................... 12
- complete_cumulative_columns ............................ 12
- CountryDataClass ......................................... 13
- Covid19DataHub .......................................... 14
- csv_reader ............................................. 16
- Cuba .................................................... 16
- DataClass ............................................... 18
- download_excel .......................................... 23
- ECDC .................................................... 23
- expect_clean_cols ....................................... 25
- expect_columns_contain_data ............................ 25
- expect_processed_cols .................................. 26
- fill_empty_dates_with_na ................................ 26
- France .................................................. 27
- france_codes ........................................... 28
- Germany ............................................... 29
- get_available_datasets .................................. 30
get_national_data ........................................ 31
get_regional_data ........................................ 33
glue_level .................................................. 35
Google .......................................................... 36
India ............................................................. 38
initialise_dataclass ......................................... 40
Italy ............................................................. 42
JHU ............................................................... 43
JHU_codes ...................................................... 45
JRC ............................................................... 46
Lithuania ......................................................... 48
lithuania_codes .............................................. 52
make_github_workflow ....................................... 52
make_new_data_source ...................................... 53
messageVerbose ............................................. 53
Mexico .......................................................... 54
mexico_codes ............................................... 56
Netherlands .................................................... 56
process_internal ............................................ 58
region_dispatch ............................................. 59
reset_cache ................................................... 59
return_data ................................................... 60
run_default_processing_fns ................................. 60
run_optional_processing_fns ................................ 61
set_negative_values_to_zero ............................... 61
SouthAfrica .................................................... 62
start_using_memoise ....................................... 63
stop_using_memoise ........................................ 63
Switzerland ..................................................... 64
test_cleaning ................................................ 65
test_download ................................................. 66
test_processing .............................................. 66
test_return ..................................................... 67
totalise_data ................................................ 67
UK ................................................................. 68
uk_codes ....................................................... 72
USA .............................................................. 73
WHO ............................................................. 74

Index  .......................................................... 76
**add_extra_na_cols**  
*Add extra columns filled with NA values to a dataset.*

**Description**  
Adds extra columns filled with NAs to a dataset. This ensures that all datasets from the covidregionaldata package return datasets of the same underlying structure (i.e. same columns).

**Usage**  
```
add_extra_na_cols(data)
```

**Arguments**  
- **data**  
  A data frame

**Value**  
A tibble with relevant NA columns added

**See Also**  
Compulsory processing functions `calculate_columns_from_existing_data()`, `complete_cumulative_columns()`, `fill_empty_dates_with_na()`

---

**all_country_data**  
*Table of available datasets along with level and other information. Rendered from the individual R6 class objects included in this package.*

**Description**  
Available datasets

**Usage**  
```
all_country_data
```

**Format**  
An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 22 rows and 10 columns.

**Value**  
A tibble of available datasets and related information.
**Belgium**

*Belgium Class for downloading, cleaning and processing notification data*

---

**Description**

Information for downloading, cleaning and processing COVID-19 region level 1 and 2 data for Belgium.

**Super class**

`covidregionaldata::DataClass` -> Belgium

**Public fields**

- `origin` name of origin to fetch data for
- `supported_levels` A list of supported levels.
- `supported_region_names` A list of region names in order of level.
- `supported_region_codes` A list of region codes in order of level. ISO 3166-2 codes are used for both region and province levels in Belgium, and for provinces these are marked as being `iso_3166_2_province`
- `common_data_urls` List of named links to raw data that are common across levels.
- `level_data_urls` List of named links to raw data specific to each level of regions. For Belgium, there are only additional data for level 1 regions.
- `source_data_cols` existing columns within the raw data
- `source_text` Plain text description of the source of the data
- `source_url` Website address for explanation/introduction of the data

**Methods**

**Public methods:**

- `Belgium$set_region_codes()`
- `Belgium$download()`
- `Belgium$clean_level_1()`
- `Belgium$clean_level_2()`
- `Belgium$clone()`

**Method** `set_region_codes()`: Set up a table of region codes for clean data

*Usage:*

`Belgium$set_region_codes()`

**Method** `download()`: Downloads data from source and (for Belgium) applies an initial data patch.
Brazil

Usage:
Belgium$download()

Method clean_level_1(): Region-level Data Cleaning
Usage:
Belgium$clean_level_1()

Method clean_level_2(): Province-level Data Cleaning
Usage:
Belgium$clean_level_2()

Method clone(): The objects of this class are cloneable with this method.
Usage:
Belgium$clone(deep = FALSE)
Arguments:
depth Whether to make a deep clone.

Source

https://epistat.sciensano.be/Data/COVID19BE_CASES_AGESEX.csv

See Also

Subnational data sources Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples

## Not run:
region <- Belgium$new(verbos = TRUE, steps = TRUE, get = TRUE, level = "2")
region$return()

## End(Not run)

Brazil

Brazil Class for downloading, cleaning and processing notification data

Description

Information for downloading, cleaning and processing COVID-19 region data for Brazil.

Super class

covidregionaldata::DataClass -> Brazil
Public fields

- origin: name of origin to fetch data for
- supported_levels: A list of supported levels.
- supported_region_names: A list of region names in order of level.
- supported_region_codes: A list of region codes in order of level.
- common_data_urls: List of named links to raw data. Data is available at the city level and is aggregated to provide state data.
- source_data_cols: existing columns within the raw data
- source_text: Plain text description of the source of the data
- source_url: Website address for explanation/introduction of the data

Methods

**Public methods:**

- **Brazil$set_region_codes():** Set up a table of region codes for clean data
  
  **Usage:**
  
  Brazil$set_region_codes()

- **Brazil$clean_common():** Common data cleaning for both levels
  
  **Usage:**
  
  Brazil$clean_common()

- **Brazil$clean_level_1():** State Level Data Cleaning
  
  **Usage:**
  
  Brazil$clean_level_1()

- **Brazil$clean_level_2():** City Level Data Cleaning
  
  **Usage:**
  
  Brazil$clean_level_2()

- **Brazil$clone():** The objects of this class are cloneable with this method.
  
  **Usage:**
  
  Brazil$clone(deep = FALSE)

  **Arguments:**
  
  deep: Whether to make a deep clone.
calculate_columns_from_existing_data

Source

https://github.com/wcota/covid19br

See Also

Subnational data sources Belgium, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples

```r
## Not run:
region <- Brazil$new(verbose = TRUE, steps = TRUE, get = TRUE)
region$return()
## End(Not run)
```

calculate_columns_from_existing_data

Cumulative counts from daily counts or daily counts from cumulative, dependent on which columns already exist

Description

Checks which columns are missing (cumulative/daily counts) and if one is present and the other not then calculates the second from the first.

Usage

calculate_columns_from_existing_data(data)

Arguments

data A data frame

Value

A data frame with extra columns if required

See Also

Compulsory processing functions add_extra_na_cols(), complete_cumulative_columns(), fill_empty_dates_with_na()
Canada Class containing origin specific attributes and methods

Description

Information for downloading, cleaning and processing COVID-19 region data for Canada.

Super class

covidregionaldata::DataClass -> Canada

Public fields

origin name of origin to fetch data for
supported_levels A list of supported levels.
supported_region_names A list of region names in order of level.
supported_region_codes A list of region codes in order of level.
common_data_urls List of named links to raw data that are common across levels.
source_data_cols existing columns within the raw data
source_text Plain text description of the source of the data
source_url Website address for explanation/introduction of the data

Methods

Public methods:
• Canada$set_region_codes()
• Canada$clean_common()
• Canada$clone()

Method set_region_codes(): Set up a table of region codes for clean data
Usage:
Canada$set_region_codes()

Method clean_common(): Provincial Level Data cleaning
Usage:
Canada$clean_common()
Arguments:
... pass additional arguments

Method clone(): The objects of this class are cloneable with this method.
Usage:
Canada$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.
Source

https://health-infobase.canada.ca

See Also

Subnational data sources Belgium, Brazil, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples

```r
## Not run:
region <- Canada$new(verbse = TRUE, steps = TRUE, get = TRUE)
region$return()
## End(Not run)
```

check_level

Checks a given level is supported

Description

Checks a given level is supported

Usage

check_level(level, supported_levels)

Arguments

- `level` A character string indicating the current level.
- `supported_levels` A character vector of supported levels

---

Colombia

Colombia Class for downloading, cleaning and processing notification data

Description

Information for downloading, cleaning and processing COVID-19 region data for Colombia

Super class

- `covidregionaldata::DataClass` -> Colombia
Public fields

- **origin**: name of origin to fetch data for
- **supported_levels**: A list of supported levels.
- **supported_region_names**: A list of region names in order of level.
- **supported_region_codes**: A list of region codes in order of level.
- **common_data_urls**: List of named links to raw data.
- **source_data_cols**: existing columns within the raw data
- **source_text**: Plain text description of the source of the data
- **source_url**: Website address for explanation/introduction of the data

Methods

**Public methods:**

- `Colombia$set_region_codes()`
- `Colombia$clean_common()`
- `Colombia$clone()`

**Method** `set_region_codes()`: Set up a table of region codes for clean data

**Usage:**

`Colombia$set_region_codes()`

**Method** `clean_common()`: Colombia specific state level data cleaning

**Usage:**

`Colombia$clean_common()`

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

**Usage:**

`Colombia$clone(deep = FALSE)`

**Arguments:**

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

Source

[https://github.com/danielcs88/colombia_covid-19/](https://github.com/danielcs88/colombia_covid-19/)

See Also

Subnational data sources [Belgium, Brazil, Canada, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA](https://github.com/danielcs88/colombia_covid-19/)

Examples

```r
## Not run:
region <- Colombia$new(quiet = TRUE, steps = TRUE, get = TRUE)
region$return()

## End(Not run)
```
**colombia_codes**  
Region Codes for Colombia Dataset.

**Description**  
The region codes for Colombia

**Usage**  
colombia_codes

**Format**  
An object of class `data.frame` with 33 rows and 2 columns.

**Value**  
A tibble of region codes and related information.

**complete_cumulative_columns**  
Completes cumulative columns if rows were added with NAs.

**Description**  
If a dataset had a row of NAs added to it (using `fill_empty_dates_with_na`) then cumulative data columns will have NAs which can cause issues later. This function fills these values with the previous non-NA value.

**Usage**  
`complete_cumulative_columns(data)`

**Arguments**  
- **data**  
  A data frame

**Value**  
A data tibble with NAs filled in for cumulative data columns.

**See Also**  
Compulsory processing functions `add_extra_na_cols()`, `calculate_columns_from_existing_data()`, `fill_empty_dates_with_na()`
CountryDataClass

R6 Class containing national level methods

Description

Acts as parent class for national data classes, allowing them to access general methods defined in DataClass() but with additional

Details

On top of the methods documented in DataClass(), this class implements a custom filter function that supports partial matching to English country names using the countrycode package.

Super class

covidregionaldata::DataClass <- CountryDataClass

Public fields

filter_level Character The level of the data to filter at. Defaults to the country level of the data.

Methods

Public methods:

• CountryDataClass$filter()
• CountryDataClass$clone()

Method filter(): Filter method for country level data. Uses countryname to match input countries with known names.

Usage:
CountryDataClass$filter(countries, level)

Arguments:
countries A character vector of target countries. Overrides the current class setting for target_regions.

level Character The level of the data to filter at. Defaults to the country level if not specified.

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

Usage:
CountryDataClass$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also

Data interface functions DataClass, get_available_datasets(), get_national_data(), get_regional_data(), initialise_dataclass()
Covid19DataHub

### Description

Attributes and methods for COVID-19 data provided by the Covid19 Data Hub

### Details

This dataset supports both national and subnational data sources with national level data returned by default. National data is sourced from John Hopkins University and so we recommend using the JHU class included in this package. Subnational data is supported for a subset of countries which can be found after cleaning using the `available_regions()` method, see the examples for more details. These data sets are minimally cleaned data files hosted by the team at COVID19 Data Hub so please see their source repository for further details (https://github.com/covid19datahub/COVID19/#data-sources) If using for analysis checking the source for further details is strongly advised.


### Super classes

```
covidregionaldata::DataClass -> covidregionaldata::CountryDataClass -> Covid19DataHub
```

### Public fields

- `origin`  name of country to fetch data for
- `supported_levels`  A list of supported levels.
- `supported_region_names`  A list of region names in order of level.
- `supported_region_codes`  A list of region codes in order of level.
- `level_data_urls`  List of named links to raw data. The first, and only entry, is be named main.
- `source_data_cols`  existing columns within the raw data
- `source_text`  Plain text description of the source of the data
- `source_url`  Website address for explanation/introduction of the data

### Methods

**Public methods:**

- `Covid19DataHub$clean_common()`
- `Covid19DataHub$clone()`

**Method** `clean_common()`: Covid19 Data Hub specific data cleaning. This takes all the raw data, renames some columns and checks types.

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

Usage:
Covid19DataHub$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Source

https://covid19datahub.io/articles/data.html

See Also

Aggregated data sources Google, JHU
National data sources ECDC, Google, JHU, JRC, WHO
Subnational data sources Belgium, Brazil, Canada, Colombia, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples

# nolint start
## Not run:
# set up a data cache
start_using_memoise()

# get all countries data
cv19dh <- Covid19DataHub$new(level = "1", get = TRUE)
cv19dh$return()

# show available regions with data at the second level of interest
cv19dh_level_2 <- Covid19DataHub$new(level = "2")
cv19dh_level_2$download()
cv19dh_level_2$clean()
cv19dh$available_regions()

# get all region data for the uk
cv19dh_level_2$filter("uk")
cv19dh_level_2$process()
cv19dh_level_2$return()

# get all regional data for the UK
uk <- Covid19DataHub$new(regions = "uk", level = "2", get = TRUE)
uk$return()

# get all subregional data for the UK
uk <- Covid19DataHub$new(regions = "uk", level = "3", get = TRUE)
uk$return()

## End(Not run)
# nolint end
### csv_reader

*Custom CSV reading function*

**Description**

Checks for use of memoise and then uses vroom::vroom.

**Usage**

```r
csv_reader(file, verbose = FALSE, guess_max = 1000, ...)
```

**Arguments**

- **file**: A URL or filepath to a CSV
- **verbose**: Logical, defaults to `TRUE`. Should verbose processing messages and warnings be returned.
- **guess_max**: Maximum number of records to use for guessing column types. Defaults to a 1000.
- **...**: extra parameters to be passed to vroom::vroom

**Value**

A data table

---

### Cuba

*Cuba Class for downloading, cleaning and processing notification data*

**Description**

Information for downloading, cleaning and processing COVID-19 region data for Cuba

**Super class**

`covidregionaldata::DataClass` $\rightarrow$ Cuba

**Public fields**

- **origin**: name of origin to fetch data for
- **supported_levels**: A list of supported levels.
- **supported_region_names**: A list of region names in order of level.
- **supported_region_codes**: A list of region codes in order of level.
- **common_data_urls**: List of named links to raw data.
- **source_data_cols**: existing columns within the raw data
- **source_text**: Plain text description of the source of the data
- **source_url**: Website address for explanation/introduction of the data
Methods

Public methods:

- Cuba$set_region_codes()
- Cuba$clean_common()
- Cuba$clone()

Method set_region_codes(): Set up a table of region codes for clean data

Usage:
Cuba$set_region_codes()

Method clean_common(): Cuba specific state level data cleaning

Usage:
Cuba$clean_common()

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

Usage:
Cuba$clone(deep = FALSE)

Arguments:

deep  Whether to make a deep clone.

Source

https://covid19cubadata.github.io/

See Also

Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples

```r
## Not run:
region <- Cuba$new(verbos = TRUE, steps = TRUE, get = TRUE)
region$return()

## End(Not run)
```
DataClass

R6 Class containing non-dataset specific methods

Description

A parent class containing non-dataset specific methods.

Details

All data sets have shared methods for extracting geographic codes, downloading, processing, and returning data. These functions are contained within this parent class and so are accessible by all data sets which inherit from here. Individual data sets can overwrite any functions or fields providing they define a method with the same name, and can be extended with additional functionality. See the individual method documentation for further details.

Public fields

- **origin**: the origin of the data source. For regional data sources this will usually be the name of the country.
- **data**: once initialised, a list of named data frames: raw (list of named raw data frames) clean (cleaned data) and processed (processed data). Data is accessed using $data.
- **supported_levels**: a list of supported levels.
- **supported_region_names**: a list of region names in order of level.
- **supported_region_codes**: a list of region codes in order of level.
- **region_name**: string, Name for the region column, e.g. 'region'. This field is filled at initialisation with the region name for the specified level (supported_region_names$level).
- **code_name**: string, Name for the codes column, e.g. 'iso_3166_2' Filled at initialisation with the code name associated with the requested level (supported_region_codes$level).
- **codes_lookup**: string or tibble, Region codes for the target origin filled by origin specific codes in set_region_codes()
- **data_urls**: List of named common and shared url links to raw data. Prefers shared if there is a name conflict.
- **common_data_urls**: List of named links to raw data that are common across levels. The first entry should be named main.
- **level_data_urls**: List of named links to raw data that are level specific. Any urls that share a name with a url from common_data_urls will be selected preferentially. Each top level list should be named after a supported level.
- **source_data_cols**: existing columns within the raw data
- **level**: target region level. This field is filled at initialisation using user inputs or defaults in $new()
- **data_name**: string, The country name followed by the level. E.g. "Italy at level 1"
- **totals**: boolean. If TRUE, returns totalled data per region up to today's date. This field is filled at initialisation using user inputs or defaults in $new()
localise  Boolean. Should region names be localised. This field is filled at initialisation using user inputs or defaults in $new()

verbose  Boolean. Display information at various stages. This field is filled at initialisation. using user inputs or defaults in $new()

steps  Boolean. Keep data from each processing step. This field is filled at initialisation. using user inputs or defaults in $new()

target_regions  A character vector of regions to filter for. Used by the filter method.

process_fns  array, additional, user supplied functions to process the data.

filter_level  Character The level of the data to filter at. Defaults to the target level.

Methods

Public methods:

- `DataClass$set_region_codes()`
- `DataClass$new()`
- `DataClass$download()`
- `DataClass$clean()`
- `DataClass$clean_common()`
- `DataClass$available_regions()`
- `DataClass$filter()`
- `DataClass$process()`
- `DataClass$get()`
- `DataClass$return()`
- `DataClass$summary()`
- `DataClass$test()`
- `DataClass$clone()`

Method `set_region_codes()`: Place holder for custom country specific function to load region codes.

Usage:

```r
DataClass$set_region_codes()
```

Method `new()`: Initialize function used by all DataClass objects. Set up the DataClass class with attributes set to input parameters. Should only be called by a DataClass class object.

Usage:

```r
DataClass$new(
  level = "1",
  filter_level,
  regions,
  totals = FALSE,
  localise = TRUE,
  verbose = TRUE,
  steps = FALSE,
  get = FALSE,
  process_fns
)```
**Arguments:**

- `level` A character string indicating the target administrative level of the data with the default being "1". Currently supported options are level 1 ("1") and level 2 ("2").
- `filter_level` A character string indicating the level to filter at. Defaults to the level of the data if not specified and if not otherwise defined in the class. Use `get_available_datasets()` for supported options by dataset.
- `regions` A character vector of target regions to be assigned to the `target_regions` field if present.
- `totals` Logical, defaults to FALSE. If TRUE, returns totalled data per region up to today’s date. If FALSE, returns the full dataset stratified by date and region.
- `localise` Logical, defaults to TRUE. Should region names be localised.
- `verbose` Logical, defaults to TRUE. Should verbose processing steps be kept and output in a list.
- `get` Logical, defaults to FALSE. Should the class `get` method be called (this will download, clean, and process data at initialisation).
- `process_fns` Array, additional functions to process the data. Users can supply their own functions here which would act on clean data and they will be called alongside our default processing functions. The default optional function added is `set_negative_values_to_zero` if `process_fns` is not set (see `process_fns` field for all defaults). If you want to keep this when supplying your own processing functions remember to add it to your list also. If you feel you have created a cool processing function that others could benefit from please submit a Pull Request to our [github repository](https://github.com) and we will consider adding it to the package.

**Method** `download()`: Download raw data from `data_urls`, stores a named list of the `data_url` name and the corresponding raw data table in `data$raw`

*Usage:*

`DataClass$download()`

**Method** `clean()`: Cleans raw data (corrects format, converts column types, etc). Works on raw data and so should be called after `download()` Calls the specific class specific cleaning method (`clean_common`) followed by level specific cleaning methods. `clean_level_[1/2]`. Cleaned data is stored in `data$clean`

*Usage:*

`DataClass$clean()`

**Method** `clean_common()`: Cleaning methods that are common across a class. By default this method is empty as if any code is required it should be defined in a child class specific `clean_common` method.

*Usage:*

`DataClass$clean_common()`

**Method** `available_regions()`: Show regions that are available to be used for filtering operations. Can only be called once `clean()` has been called. Filtering level is determined by checking the `filter_level` field.

*Usage:*)
**Method** `filter()`: Filter cleaned data for a specific region To be called after `clean()`

**Usage:**

```
DataClass$filter(regions, level)
```

**Arguments:**
- `regions` A character vector of target regions. Overrides the current class setting for `target_regions`.
- `level` Character The level of the data to filter at. Defaults to the lowest level in the data.

**Method** `process()`: Processes data by adding and calculating absent columns. Called on clean data (after `clean()`). Some countries may have data as new events (e.g. number of new cases for that day) whilst others have a running total up to that date. Processing calculates these based on what the data comes with via the functions `region_dispatch()` and `process_internal()`, which does the following:
- Adds columns not present in the data `add_extra_na_cols()`
- Ensures there are no negative values `set_negative_values_to_zero()`
- Removes NA dates `fill_empty_dates_with_na()`
- Calculates cumulative data `complete_cumulative_columns()`
- Calculates missing columns from existing ones `calculate_columns_from_existing_data()`

**Usage:**

```
DataClass$process(process_fns)
```

**Arguments:**
- `process_fns` Array, additional functions to process the data. Users can supply their own functions here which would act on clean data and they will be called alongside our default processing functions. The default optional function added is `set_negative_values_to_zero`. if `process_fns` is not set (see `process_fns` field for all defaults).

**Method** `get()`: Get data related to the data class. This runs each distinct step in the workflow in order. Internally calls `download()`, `clean()`, `filter()` and `process()` download, clean, filter and process methods.

**Usage:**

```
DataClass$get()
```

**Method** `return()`: Return data. Designed to be called after `process()` this uses the steps argument to return either a list of all the data preserved at each step or just the processed data. For most datasets a custom method should not be needed.

**Usage:**

```
DataClass$return()
```

**Method** `summary()`: Create a table of summary information for the data set being processed.

**Usage:**
Returns: Returns a single row summary tibble containing the origin of the data source, class, level 1 and 2 region names, the type of data, the urls of the raw data and the columns present in the raw data.

Method test(): Run tests on a country class instance. Calling test() on a class instance runs tests with the settings in use. For example, if you set level = "1" and localise = FALSE the tests will be run on level 1 data which is not localised. Rather than downloading data for a test users can provide a path to a snapshot file of data to test instead. Tests are run on a clone of the class. This method calls generic tests for all country class objects. It also calls country specific tests which can be defined in an individual country class method called specific_tests(). The snapshots contain the first 1000 rows of data. For more details see the 'testing' vignette: vignette(testing).

Usage:
DataClass$test(
  download = FALSE,
  snapshot_dir = paste0(tempdir(), "/snapshots"),
  all = FALSE,
  ...
)

Arguments:
download logical. To download the data (TRUE) or use a snapshot (FALSE). Defaults to FALSE.

snapshot_dir character array the name of a directory to save the downloaded data or read from. If not defined a directory called 'snapshots' will be created in the temp directory. Snapshots are saved as rds files with the class name and level: e.g. Italy_level_1.rds.

all logical. Run tests with all settings (TRUE) or with those defined in the current class instance (FALSE). Defaults to FALSE.

... Additional parameters to pass to specific_tests

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

Usage:
DataClass$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also
Data interface functions CountryDataClass, get_available_datasets(), get_national_data(), get_regional_data(), initialise_dataclass()
download_excel

Description

Download Excel Documents

Usage

download_excel(url, archive, verbose = FALSE, transpose = TRUE, ...)

Arguments

url
- Character string containing the full URL to the Excel document.

archive
- Character string naming the file name to assign in the temporary directory.

verbose
- Logical, defaults to TRUE. Should verbose processing messages and warnings be returned.

transpose
- Logical, should the read in data be transposed

...
- Additional parameters to pass to read_excel().

Value

A data.frame.

ECDC

R6 Class containing specific attributes and methods for the European Centre for Disease Prevention and Control dataset

Description

Information for downloading, cleaning and processing the European Centre for Disease Prevention and Control COVID-19 data.

Super classes

covidregionaldata::DataClass -> covidregionaldata::CountryDataClass -> ECDC
Public fields

origin  name of origin to fetch data for
supported_levels  A list of supported levels.
supported_region_names  A list of region names in order of level.
supported_region_codes  A list of region codes in order of level.
common_data_urls  List of named links to raw data.
source_data_cols  existing columns within the raw data
source_text  Plain text description of the source of the data
source_url  Website address for explanation/introduction of the data

Methods

Public methods:

• ECDC$clean_common()
• ECDC$return()
• ECDC$specific_tests()
• ECDC$clone()

Method clean_common(): ECDC specific state level data cleaning

Usage:
ECDC$clean_common()

Method return(): Specific return settings for the ECDC dataset.

Usage:
ECDC$return()

Method specific_tests(): Run additional tests on ECDC class. Tests ECDC has required additional columns and that there is only one row per country. Designed to be run from test and not run directly.

Usage:
ECDC$specific_tests(self_copy, ...)

Arguments:
self_copy  R6class the object to test
...  Extra params passed to specific download functions

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

Usage:
ECDC$clone(deep = FALSE)

Arguments:
depth  Whether to make a deep clone.

Source

expect_clean_cols

**See Also**

National data sources Covid19DataHub, Google, JHU, JRC, WHO

**Examples**

```r
### Not run: national <- ECDC$new( verbose = TRUE, steps = TRUE, get = TRUE) national$return()
### End(Not run)
```

---

**expect_clean_cols**  
*Test clean columns contain the correct data and types*

**Description**

Checks the date column is an s3 class and that region level column is a character in the cleaned data (data$clean)

**Usage**

```r
expect_clean_cols(data, level)
```

**Arguments**

- **data**: The clean data to check
- **level**: character_array the level of the data to check

**See Also**

Functions used for testing data is cleaned and processed correctly *expect_columns_contain_data()*, *expect_processed_cols()*, *test_cleaning()*, *test_download()*, *test_processing()*, *test_return()*

---

**expect_columns_contain_data**  
*Test that cleaned columns contain data/*

**Description**

Checks that cleaned columns cases, deaths, recovered and test (new and total) are not entirely composed of NAs.

**Usage**

```r
expect_columns_contain_data(DataClass_obj)
```
Arguments

DataClass_obj The DataClass object (R6Class) to perform checks on. Must be a DataClass or DataClass child object.

See Also

Functions used for testing data is cleaned and processed correctly expect_clean_cols(), expect_processed_cols(), test_cleaning(), test_download(), test_processing(), test_return()

---

expect_processed_cols Test that processed columns contain the correct data and types

Description

Checks that processed data columns date, cases_new, cases_total, deaths_new, deaths_total and that region level have the correct types.

Usage

expect_processed_cols(data, level = "1", localised = TRUE)

Arguments

data The data to check
level character_array the level of the data to check
localised logical to check localised data or not, defaults to TRUE.

See Also

Functions used for testing data is cleaned and processed correctly expect_clean_cols(), expect_columns_contain_data(), test_cleaning(), test_download(), test_processing(), test_return()

---

fill_empty_dates_with_na

Add rows of NAs for dates where a region does not have any data

Description

There are points, particularly early during data collection, where data was not collected for all regions. This function finds dates which have data for some regions, but not all, and adds rows of NAs for the missing regions. This is mainly for reasons of completeness.

Usage

fill_empty_dates_with_na(data)
Arguments
data A data frame

Value
A tibble with rows of NAs added.

See Also
Compulsory processing functions `add_extra_na_cols()`, `calculate_columns_from_existing_data()`, `complete_cumulative_columns()`.

France
France Class containing origin specific attributes and methods

Description
Information for downloading, cleaning and processing COVID-19 region data for France.

Super class
covidregionaldata::DataClass -> France

Public fields
origin name of origin to fetch data for
supported_levels A list of supported levels.
supported_region_names A list of region names in order of level.
supported_region_codes A list of region codes in order of level.
level_data_urls List of named links to raw data that are level specific.
source_data_cols existing columns within the raw data
source_text Plain text description of the source of the data
source_url Website address for explanation/introduction of the data

Methods
Public methods:
- `France$set_region_codes()`
- `France$clean_level_1()`
- `France$clean_level_2()`
- `France$clone()`

Method `set_region_codes()`: Set up a table of region codes for clean data

Usage:
France$set_region_codes()

**Method** clean_level_1(): Region Level Data Cleaning

**Usage:**
France$clean_level_1()

**Method** clean_level_2(): Department Level Data Cleaning

**Usage:**
France$clean_level_2()

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

**Usage:**
France$clone(deep = FALSE)

**Arguments:**
depth  Whether to make a deep clone.

**Source**
https://www.data.gouv.fr/fr/datasets/r/406c6a23-e283-4300-9484-54e78c8ae675
https://www.data.gouv.fr/fr/datasets/r/6fadff46-9efd-4c53-942a-54aca783c30c
https://www.data.gouv.fr/fr/datasets/r/001aca18-df6a-45c8-89e6-f82d689e6c01

**See Also**
Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

**Examples**

```r
## Not run:
region <- France$new(level = "2", verbose = TRUE, steps = TRUE, get = TRUE)
region$return()

## End(Not run)
```

---

**france_codes**

*Region Codes for France Dataset.*

**Description**
The region codes for France

**Usage**

france_codes
Format

An object of class `data.frame` with 104 rows and 5 columns.

Value

A tibble of region codes and related information.

---

<table>
<thead>
<tr>
<th>Germany</th>
<th>Germany Class for downloading, cleaning and processing notification data</th>
</tr>
</thead>
</table>

Description

Information for downloading, cleaning and processing COVID-19 region level 1 and 2 data for Germany.

Super class

covidregionaldata::DataClass -> Germany

Public fields

- `origin` name of origin to fetch data for
- `supported_levels` A list of supported levels.
- `supported_region_names` A list of region names in order of level.
- `supported_region_codes` A list of region codes in order of level.
- `common_data_urls` List of named links to raw data. The first, and only entry, is be named main.
- `source_data_cols` existing columns within the raw data
- `source_text` Plain text description of the source of the data
- `source_url` Website address for explanation/introduction of the data

Methods

**Public methods:**

- `Germany$set_region_codes()`
- `Germany$clean_common()`
- `Germany$clean_level_1()`
- `Germany$clean_level_2()`
- `Germany$clone()`

**Method `set_region_codes()`**: Set up a table of region codes for clean data

*Usage:*

`Germany$set_region_codes()`
Method `clean_common()`: Common Data Cleaning
   Usage:
   Germany$clean_common()

Method `clean_level_1()`: Bundesland Level Data Cleaning
   Usage:
   Germany$clean_level_1()

Method `clean_level_2()`: Landkreis Level Data Cleaning
   Usage:
   Germany$clean_level_2()

Method `clone()`: The objects of this class are cloneable with this method.
   Usage:
   Germany$clone(deep = FALSE)
   Arguments:
   deep  Whether to make a deep clone.

Source
   https://opendata.arcgis.com/datasets/dd4580c810204019a7b8eb3e0b329dd6_0.csv

See Also
   Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples
   ```
   ## Not run:
   region <- Germany$new(verbos = TRUE, steps = TRUE, level = "2", get = TRUE)
   region$return()
   
   ## End(Not run)
   ```

get_available_datasets
   Get supported data sets

Description
   Returns data on what countries are available from the data provided with this package either using a cached dataset or built by searching the target namespace.
get_national_data

Usage

get_available_datasets(type, render = FALSE, namespace = "covidregionaldata")

Arguments

type A character vector indicating the types of data to return. Current options include "national" (which are datasets at the national level which inherit from CountryDataClass) and "regional" (which are datasets at the regional level which inherit directly from DataClass()).

render Logical If TRUE the supported data set table is built from the available classes using summary methods. If FALSE the supported data set table is taken from package data. Defaults to FALSE.

namespace Character string The name of the namespace to search for class objects. Defaults to "covidregionaldata" as the package.

Value

A list of available data sets and the spatial aggregation data is available for.

See Also

Data interface functions CountryDataClass, DataClass, get_national_data(), get_regional_data(), initialise_dataclass()

Examples

# see all available datasets
get_available_datasets()

# see only national level datasets
get_available_datasets("national")

# see only regional level datasets
get_available_datasets("regional")

# render the data
get_available_datasets(render = TRUE)

get_national_data Get national-level data for countries globally from a range of sources

Description

Provides an interface to source specific classes which support national level data. For simple use cases this allows downloading clean, standardised, national-level COVID-19 data sets. Internally this uses the CountryDataClass() parent class which allows documented downloading, cleaning, and processing. Optionally all steps of data processing can be returned along with the functions used for processing but by default just the finalised processed data is returned. See the examples for some potential use cases and the links to lower level functions for more details and options.
get_national_data

Usage

get_national_data(
  countries,  # A character vector specifying country names of interest. Used to filter the data.
  source = "who",  # A character string specifying the data source (not case dependent). Defaults to WHO (the World Health Organisation). See get_available_datasets("national") for all options.
  level = "1",  # A character string indicating the target administrative level of the data with the default being "1". Currently supported options are level 1 ("1") and level 2 ("2"). Use get_available_datasets() for supported options by dataset.
  totals = FALSE,  # Logical, defaults to FALSE. If TRUE, returns totalled data per region up to today's date. If FALSE, returns the full dataset stratified by date and region.
  steps = FALSE,  # Logical, defaults to TRUE. Should all processing and cleaning steps be kept and output in a list.
  class = FALSE,  # Logical, defaults to FALSE. If TRUE returns the DataClass object rather than a tibble or a list of tibbles. Overrides steps.
  verbose = TRUE,  # Logical, defaults to TRUE. Should verbose processing messages and warnings be returned.
  country = deprecated(),  # A character string specifying a country to filter for.
  ...  # Additional arguments to pass to class specific functionality.
)

Arguments

countries  # A character vector specifying country names of interest. Used to filter the data.
source  # A character string specifying the data source (not case dependent). Defaults to WHO (the World Health Organisation). See get_available_datasets("national") for all options.
level  # A character string indicating the target administrative level of the data with the default being "1". Currently supported options are level 1 ("1") and level 2 ("2"). Use get_available_datasets() for supported options by dataset.
totals  # Logical, defaults to FALSE. If TRUE, returns totalled data per region up to today's date. If FALSE, returns the full dataset stratified by date and region.
steps  # Logical, defaults to FALSE. Should all processing and cleaning steps be kept and output in a list.
class  # Logical, defaults to FALSE. If TRUE returns the DataClass object rather than a tibble or a list of tibbles. Overrides steps.
verbose  # Logical, defaults to TRUE. Should verbose processing messages and warnings be returned.
country  # [Deprecated] A character string specifying a country to filter for.
...

Value

A tibble with data related to cases, deaths, hospitalisations, recoveries and testing.

See Also

WHO(), ECDC(), JHU(), Google()

Data interface functions CountryDataClass, DataClass, get_available_datasets(), get_regional_data(), initialise_dataclass()
### Examples

```
## Not run:
# set up a data cache
start_using_memoise()

# download all national data from the WHO
get_national_data(source = "who")

# download data for Canada keeping all processing steps
get_national_data(countries = "canada", source = "ecdc")

# download data for Canada from the JHU and return the full class
jhu <- get_national_data(countries = "canada", source = "jhu", class = TRUE)
jhu

# return the JHU data for canada
jhu$return()

# check which regions the JHU supports national data for
jhu$available_regions()

# filter instead for France (and then reprocess)
jhu$filter("France")
jhu$process()

# explore the structure of the stored JHU data
jhu$data

## End(Not run)
```

---

### Description

Provides an interface to source specific classes which support regional level data. For simple use cases this allows downloading clean, standardised, regional-level COVID-19 data sets. Internally this uses the `DataClass()` parent class which allows documented downloading, cleaning, and processing. Optionally all steps of data processing can be returned along with the functions used for processing but by default just the finalised processed data is returned. See the examples for some potential use cases and the links to lower level functions for more details and options.

### Usage

```
get_regional_data(
  country,
  level = "1",
  totals = FALSE,
  localise = TRUE,
```
get_regional_data

  steps = FALSE,
  class = FALSE,
  verbose = TRUE,
  regions,
  include_level_2_regions = deprecated(),
  localise_regions = deprecated(),
  ...
)

Arguments

country A character string specifying the country to get data from. Not case dependent. Name should be the English name. For a list of options use get_available_datasets().

level A character string indicating the target administrative level of the data with the default being "1". Currently supported options are level 1 ("1") and level 2 ("2"). Use get_available_datasets() for supported options by dataset.

totals Logical, defaults to FALSE. If TRUE, returns totalled data per region up to today’s date. If FALSE, returns the full dataset stratified by date and region.

localise Logical, defaults to TRUE. Should region names be localised.

steps Logical, defaults to FALSE. Should all processing and cleaning steps be kept and output in a list.

class Logical, defaults to FALSE. If TRUE returns the DataClass object rather than a tibble or a list of tibbles. Overrides steps.

verbose Logical, defaults to TRUE. Should verbose processing messages and warnings be returned.

regions A character vector of target regions to be assigned to the target_regions field and used to filter the returned data.

include_level_2_regions [Deprecated] Boolean. If TRUE, returns data stratified by level 2 regions. If FALSE, stratified by Level 1. Note that Level 2 region data is not always available. In these cases the user will get a warning and the Level 1 data will be returned.

localise_regions [Deprecated] Logical, defaults to TRUE. Should region names be localised.

... Additional arguments to pass to class specific functionality.

Value

A tibble with data related to cases, deaths, hospitalisations, recoveries and testing stratified by regions within the given country.

See Also

Italy(), UK()

Data interface functions CountryDataClass, DataClass, get_available_datasets(), get_national_data(), initialise_dataclass()
Examples

```r
## Not run:
# set up a data cache
start_using_memoise()

# download data for Italy
get_regional_data("italy")

# return totals for Italy with no localisation
get_regional_data("italy", localise = FALSE, totals = TRUE)

# download data for the UK but return the class
uk <- get_regional_data("United Kingdom", class = TRUE)
uk

# return UK data from the class object]
uk$return()

## End(Not run)
```

---

**glue_level**  
*Glue the spatial level into a variable name*

---

**Description**

Glue the spatial level into a variable name

**Usage**

```r
glue_level(level)
```

**Arguments**

- `level`  
  A character string indicating the current level.

**Value**

A string in the form "level_1_region".
Google

R6 Class containing specific attributes and methods for Google data

Description

Google specific information for downloading, cleaning and processing covid-19 region data for an example Country. The function works the same as other national data sources, however, data from Google supports three subregions (country, subregion and subregion2) which can be accessed using the 'level' argument. There is also more data available, such as hospitalisations data. The raw data comes as three separate data sets, "epidemiology" which is comprised of cases, tests and deaths, "index", which holds information about countries linking the other data sets, and "hospitalizations" which holds data about number of people in hospital, ICU, etc.

Super classes

covidregionaldata::DataClass -> covidregionaldata::CountryDataClass -> Google

Public fields

origin  name of country to fetch data for
supported_levels  A list of supported levels.
supported_region_names  A list of region names in order of level.
supported_region_codes  A list of region codes in order of level.
common_data_urls  List of named links to raw data.
source_data_cols  existing columns within the raw data
source_text  Plain text description of the source of the data
source_url  Website address for explanation/introduction of the data

Methods

Public methods:

• Google$clean_common()
• Google$clean_level_1()
• Google$clean_level_2()
• Google$new()
• Google$clone()

Method clean_common(): GoogleData specific subregion2 level data cleaning. This takes all the raw data, puts into a single data frame, renames some columns and checks types.

Usage:

Google$clean_common()

Method clean_level_1():  Google specific subregion level data cleaning. Takes the data cleaned by clean_common and aggregates it to the country level (level 1).
Usage:
Google$clean_level_1()

Method clean_level_2():  Google specific subregion2 level data cleaning. Takes the data cleaned by clean_common and aggregates it to the subregion level (level 2).
  Usage:
Google$clean_level_2()

Method new(): custom initialize for Google
  Usage:
Google$new(...)
  Arguments:
...  arguments to be passed to DataClass and initialize Google

Method clone(): The objects of this class are cloneable with this method.
  Usage:
Google$clone(deep = FALSE)
  Arguments:
  deep  Whether to make a deep clone.

Source

See Also
  Aggregated data sources Covid19DataHub, JHU
  National data sources Covid19DataHub, ECDC, JHU, JRC, WHO
  Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples
  # nolint start
  ## Not run:
  # set up a data cache
  start_using_memoise()

  # get all countries
  national <- Google$new(level = "1", get = TRUE)
national$return()

  # show available regions with data at the second level of interest
  google_level_2 <- Google$new(level = "2")
google_level_2$download()
google_level_2$clean()
google$available_regions()
India

# get all region data for the uk
google_level_2$filter("uk")
google_level_2$process()
google_level_2$return()

# get all regional data for the UK
uk <- Google$new(regions = "uk", level = "2", get = TRUE)
uk$return()

# get all subregional data for the UK
uk <- Google$new(regions = "uk", level = "3", get = TRUE)
uk$return()

## End(Not run)
# nolint end

---

**India**

*India Class for downloading, cleaning and processing notification data*

---

**Description**

Information for downloading, cleaning and processing COVID-19 region data for India.

**Super class**

`covidregionaldata::DataClass` -> India

**Public fields**

- `origin` name of origin to fetch data for
- `supported_levels` A list of supported levels.
- `supported_region_names` A list of region names in order of level.
- `supported_region_codes` A list of region codes in order of level.
- `common_data_urls` List of named links to raw data.
- `source_data_cols` existing columns within the raw data
- `source_text` Plain text description of the source of the data
- `source_url` Website address for explanation/introduction of the data

**Methods**

**Public methods:**

- `India$set_region_codes()`
- `India$clean_common()`
- `India$get_desired_status()`
• India$clone()

**Method** `set_region_codes()`: Set up a table of region codes for clean data

*Usage:*
India$set_region_codes()

**Method** `clean_common()`: India state level data cleaning

*Usage:*
India$clean_common()

**Method** `get_desired_status()`: Extract data from raw table

*Usage:*
India$get_desired_status(status)

*Arguments:*
status The data to extract

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

*Usage:*
India$clone(deep = FALSE)

*Arguments:*
deep Whether to make a deep clone.

**Source**

https://www.covid19india.org

**See Also**

Subnational data sources **Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA**

**Examples**

```r
## Not run:
region <- India$new(verbse = TRUE, steps = TRUE, get = TRUE)
region$return()

## End(Not run)
```
initialise_dataclass  *Initialise a child class of DataClass if it exists*

**Description**

This function initialises classes based on the `DataClass()` which allows documented downloading, cleaning, and processing. See the examples for some potential use cases and the `DataClass()` documentation for more details.

**Usage**

```r
initialise_dataclass(
  class = character(),
  level = "1",
  totals = FALSE,
  localise = TRUE,
  regions,
  verbose = TRUE,
  steps = FALSE,
  get = FALSE,
  type = c("national", "regional"),
  ...
)
```

**Arguments**

- **class**: A character string specifying the `DataClass()` to initialise. Not case dependent and matching is based on either the class name or the its country definition. For a list of options use `get_available_datasets()`.
- **level**: A character string indicating the target administrative level of the data with the default being "1". Currently supported options are level 1 ("1") and level 2 ("2"). Use `get_available_datasets()` for supported options by dataset.
- **totals**: Logical, defaults to FALSE. If TRUE, returns totalled data per region up to today’s date. If FALSE, returns the full dataset stratified by date and region.
- **localise**: Logical, defaults to TRUE. Should region names be localised.
- **regions**: A character vector of target regions to be assigned to the `target_regions` field and used to filter the returned data.
- **verbose**: Logical, defaults to TRUE. Should verbose processing messages and warnings be returned.
- **steps**: Logical, defaults to FALSE. Should all processing and cleaning steps be kept and output in a list.
- **get**: Logical, defaults to FALSE. Should the class `get` method be called (this will download, clean, and process data at initialisation).
initialise_dataclass

41

type A character vector indicating the types of data to return. Current options include "national" (which are datasets at the national level which inherit from CountryDataClass) and "regional" (which are datasets at the regional level which inherit directly from DataClass).

... Additional arguments to pass to class specific functionality.

Value

An initialised version of the target class if available, e.g. Italy()

See Also

Data interface functions CountryDataClass, DataClass, get_available_datasets(), get_national_data(), get_regional_data()

Examples

## Not run:
# set up a cache to store data to avoid downloading repeatedly
start_using_memoise()

# check currently available datasets
get_available_datasets()

# initialise a data set in the United Kingdom
# at the UTLA level
utla <- UK$new(level = "2")

# download UTLA data
utla$download()

# clean UTLA data
utla$clean()

# inspect available level 1 regions
utla$available_regions(level = "1")

# filter data to the East of England
utla$filter("East of England")

# process UTLA data
utla$process()

# return processed and filtered data
utla$return()

# inspect all data steps
utla$data

# initialise Italian data, download, clean and process it
italy <- initialise_dataclass("Italy", get = TRUE)
italy$return()
# initialise ECDC data, fully process it, and return totals
ecdc <- initialise_dataclass("ecdc", get = TRUE, totals = TRUE)
ecdc$return()

## End(Not run)

### Italy

**Italy Class for downloading, cleaning and processing notification data**

**Description**

Information for downloading, cleaning and processing COVID-19 region data for Italy.

**Super class**

covidregionaldata::DataClass -> Italy

**Public fields**

- origin: name of origin to fetch data for
- supported_levels: A list of supported levels.
- supported_region_names: A list of region names in order of level.
- supported_region_codes: A list of region codes in order of level.
- common_data_urls: List of named links to raw data. The first, and only entry, is be named main.
- source_data_cols: existing columns within the raw data
- source_text: Plain text description of the source of the data
- source_url: Website address for explanation/introduction of the data

**Methods**

**Public methods:**

- Italy$set_region_codes()
- Italy$clean_common()
- Italy$clone()

**Method** set_region_codes(): Set up a table of region codes for clean data

**Usage:**

Italy$set_region_codes()

**Method** clean_common(): State level data cleaning

**Usage:**

Italy$clean_common()

**Method** clone(): The objects of this class are cloneable with this method.
Usage:
Italy$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Source
https://github.com/pcm-dpc/COVID-19/

See Also
Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples
```r
## Not run:
region <- Italy$new( verbose = TRUE, steps = TRUE, get = TRUE)
region$return()
## End(Not run)
```

JHU

_R6 Class containing specific attributes and methods for John Hopkins University data_

Description


Details

This dataset support both national and subnational data sources with national level data returned by default. Subnational data is supported for a subset of countries which can be found after cleaning using the `available_regions()` method, see the examples for more details. These data sets are sourced, cleaned, standardised by the JHU team so please see the source repository for further details. Note that unlike many other data sets this means methods applied to this source are not being applied to raw surveillance data but instead to already cleaned data. If using for analysis checking the JHU source for further details is advisable.

Super classes

    covidregionaldata::DataClass -> covidregionaldata::CountryDataClass -> JHU

Public fields

    origin  name of country to fetch data for
    supported_levels  A list of supported levels.
    supported_region_names  A list of region names in order of level.
    supported_region_codes  A list of region codes in order of level.
    common_data_urls  List of named links to raw data. The first, and only entry, is be named main.
    source_data_cols  existing columns within the raw data
    source_text  Plain text description of the source of the data
    source_url  Website address for explanation/introduction of the data

Methods

Public methods:

    • JHU$set_region_codes()
    • JHU$clean_common()
    • JHU$clean_level_1()
    • JHU$clone()

Method set_region_codes(): Set up a table of region codes for clean data

Usage:
    JHU$set_region_codes()

Method clean_common(): JHU specific data cleaning. Joins the raw data sets, checks column types and renames where needed.

Usage:
    JHU$clean_common()

Method clean_level_1(): JHU specific country level data cleaning. Aggregates the data to the country (level 2) level.

Usage:
    JHU$clean_level_1()

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

Usage:
    JHU$clone(deep = FALSE)

Arguments:
    deep  Whether to make a deep clone.

Source

https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data
JHU_codes

Description

The region codes for JHU

Usage

JHU_codes

Format

An object of class spec_tbl_df (inherits from tbl_df, tbl, data.frame) with 4193 rows and 2 columns.
Value

A tibble of region codes and related information.

Description

Class for downloading, cleaning and processing COVID-19 region data from the European Commission’s Joint Research Centre. Subnational data (admin level 1) on numbers of contagious and fatalities by COVID-19, collected directly from the National Authoritative sources (National monitoring websites, when available). For more details see https://github.com/ec-jrc/COVID-19

Super classes

covidregionaldata::DataClass -> covidregionaldata::CountryDataClass -> JRC

Public fields

origin name of origin to fetch data for

supported_levels A list of supported levels.
supported_region_names A list of region names in order of level.
supported_region_codes A list of region codes in order of level.
level_data_urls List of named links to raw data.
source_data_cols existing columns within the raw data
source_text Plain text description of the source of the data
source_url Website address for explanation/introduction of the data

Methods

Public methods:

• JRC$clean_common()
• JRC$clean_level_1()
• JRC$clean_level_2()
• JRC$clone()

Method clean_common(): JRC specific data cleaning. The raw source data columns are converted to the correct type and renamed appropriately to match the standard for general processing.

Usage:
JRC$clean_common()

Method clean_level_1(): JRC specific country level data cleaning. Selects country level (level 1) columns from the data ready for further processing.
Usage:
JRC$clean_level_1()

Method clean_level_2(): JRC specific region level data cleaning. Selects country (level 1) and region (level 2) columns from the data ready for further processing.

Usage:
JRC$clean_level_2()

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

Usage:
JRC$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Source

https://github.com/ec-jrc/COVID-19

See Also

National data sources Covid19DataHub, ECDC, Google, JHU, WHO

Examples

## Not run:
# get country level data
jrc_level_1 <- JRC$new(level = "1", verbose = TRUE, steps = TRUE, get = TRUE)  
jrc_level_1$return()

# show available regions with data at the first level of interest (country)
jrc_level_1$returnable_regions()

# get region level data
jrc_level_2 <- JRC$new(level = "2", verbose = TRUE, steps = TRUE, get = TRUE)  
jrc_level_2$return()

# show available regions with data at the second level of interest (region)
jrc_level_2$returnable_regions()

## End(Not run)
Description

Information for downloading, cleaning and processing COVID-19 region level 1 and 2 data for Lithuania.

OSP Data fields

The Official Statistics Portal (OSP) provides many data series in their table.

The full range of these vectors can be returned by setting all_osp_fields to TRUE.

The following describes the data provided by the OSP.

<table>
<thead>
<tr>
<th>field</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>the reporting day during which the events occurred or at the end of which the accounting was performed.</td>
</tr>
<tr>
<td>municipality_code</td>
<td>code of the municipality assigned to persons</td>
</tr>
<tr>
<td>municipality_name</td>
<td>the name of the municipality assigned to the persons</td>
</tr>
<tr>
<td>population</td>
<td>population size according to the data of the beginning of 2021, according to the declared place of residence.</td>
</tr>
<tr>
<td>ab_pos_day</td>
<td>Number of positive antibody test responses, days</td>
</tr>
<tr>
<td>ab_neg_day</td>
<td>Number of negative antibody test responses, days</td>
</tr>
<tr>
<td>ab_tot_day</td>
<td>Number of antibody tests, daily</td>
</tr>
<tr>
<td>ab_prc_day</td>
<td>Percentage of positive antibody test responses per day</td>
</tr>
<tr>
<td>ag_pos_day</td>
<td>Number of positive antigen test responses, daily</td>
</tr>
<tr>
<td>ag_neg_day</td>
<td>Number of negative antigen test responses, daily</td>
</tr>
<tr>
<td>ag_tot_day</td>
<td>Number of antigen tests, daily</td>
</tr>
<tr>
<td>ag_prc_day</td>
<td>Percentage of positive responses to antigen tests per day</td>
</tr>
<tr>
<td>pcr_pos_day</td>
<td>number of positive PCR test responses, daily</td>
</tr>
<tr>
<td>pcr_neg_day</td>
<td>Number of PCR test negative responses, daily</td>
</tr>
<tr>
<td>pcr_tot_day</td>
<td>number of PCR tests per day</td>
</tr>
<tr>
<td>pcr_prc_day</td>
<td>Percentage of positive PCR test responses per day</td>
</tr>
<tr>
<td>dgn_pos_day</td>
<td>Number of positive answers to diagnostic tests / tests, days</td>
</tr>
<tr>
<td>dgn_neg_day</td>
<td>Number of negative answers to diagnostic tests / tests, days</td>
</tr>
<tr>
<td>dgn_prc_day</td>
<td>Number of diagnostic examinations / tests, days</td>
</tr>
<tr>
<td>dgn_tot_day</td>
<td>Percentage of positive answers to diagnostic tests / tests per day</td>
</tr>
<tr>
<td>dgn_tot_day_gmp</td>
<td>Number of diagnostic examinations / tests of samples collected at mobile points, days</td>
</tr>
<tr>
<td>daily_deaths_def1</td>
<td>The number of new deaths per day according to the (narrowest) COVID death definition No. 1.</td>
</tr>
<tr>
<td>daily_deaths_def2</td>
<td>Number of new deaths per day according to COVID death definition No. 2.</td>
</tr>
<tr>
<td>daily_deaths_def3</td>
<td>Number of new deaths per day according to COVID death definition No. 3.</td>
</tr>
<tr>
<td>daily_deaths_all</td>
<td>Daily deaths in Lithuania (by date of death)</td>
</tr>
<tr>
<td>incidence +</td>
<td>Number of new COVID cases per day (laboratory or physician confirmed)</td>
</tr>
<tr>
<td>cumulative_totals +</td>
<td>Total number of COVID cases (laboratory or physician confirmed)</td>
</tr>
<tr>
<td>active_de_jure</td>
<td>Declared number of people with COVID</td>
</tr>
<tr>
<td>active_sttstcl</td>
<td>Statistical number of people with COVID</td>
</tr>
<tr>
<td>dead_cases</td>
<td>The number of dead persons who were ever diagnosed with COVID</td>
</tr>
</tbody>
</table>
Criteria for attributing deaths

Beginning in February 2021 the OSP publishes death counts according to three different criteria, from most to least strictly attributed to COVID-19.

1. of Number of deaths with COVID-19 (coronavirus infection) as the leading cause of death. The indicator is calculated by summing all registered records of medical form E106 (unique persons), in which the main cause of death is IPC disease codes U07.1 or U07.2. Deaths due to external causes are not included (ICD disease codes are V00-Y36, or Y85-Y87, or Y89, or S00-T79, or T89-T98).

2. with Number of deaths with COVID-19 (coronavirus infection) of any cause of death. The indicator is calculated by summing all registered records of the medical form E106 (unique persons), in which the ICD disease codes U07.1, U07.2, U07.3, U07.4, U07.5 are indicated as the main, direct, intermediate cause of death or other important pathological condition, or identified as related to COVID-19 disease (coronavirus infection). Deaths due to external causes are not included (ICD disease codes are V00-Y36, or Y85-Y87, or Y89, or S00-T79, or T89-T98).

3. after Number of deaths from any cause of COVID-19 or COVID-19 deaths due to non-external causes within 28 days. The indicator is calculated by summing all registered records of the medical form E106 (unique persons), in which the ICD disease codes U07.1, U07.2, U07.3, U07.4, U07 are indicated as the main, direct, intermediate cause of death or other important pathological condition, or identified as related to COVID-19 disease (coronavirus infection) and all records of medical form E106 (unique individuals) where the person died within the last 28 days after receiving a positive diagnostic response to the SARS-CoV-2 test or had an entry in medical form E025 with ICD disease code U07.2 or U07.1. Deaths due to external causes are not included (ICD disease codes are V00-Y36, or Y85-Y87, or Y89, or S00-T79, or T89-T98).

The number of deaths reported in the last day is preliminary and increases by about 20-40% in a few days. Such a "delay" in the data is natural: for example, for those who died last night, a death certificate is likely to be issued as soon as this report is published this morning.

De jure and statistical counts

Beginning in February 2021 the OSP makes statistical estimates of the number of recovered and active cases, since review of the data showed that some cases individuals still considered as active cases had recovered, but not documented or registered as such.
These are listed as by the OSP as active_de_jure and recovered_de_jure (officially still considered sick), and active_sttstcl and recovered_sttstcl (an estimate of how many of these are still ill).

Super class

\texttt{covidregionaldata::DataClass \rightarrow Lithuania}

Public fields

- \texttt{origin} name of origin to fetch data for
- \texttt{supported_levels} A list of supported levels.
- \texttt{supported_region_names} A list of region names in order of level.
- \texttt{supported_region_codes} A list of region codes in order of level.
- \texttt{common_data_urls} List of named links to raw data that are common across levels.
- \texttt{source_data_cols} existing columns within the raw data
- \texttt{source_text} Plain text description of the source of the data
- \texttt{source_url} Website address for explanation/introduction of the data
- \texttt{death_definition} which criteria of deaths attributed to COVID to use
- \texttt{recovered_definition} whether to use the official counts of recovered cases or the statistical estimates provided by OSP
- \texttt{all_osp_fields} whether to return all the data vectors provided by OSP
- \texttt{national_data} whether to return data rows for national results

Methods

Public methods:

- \texttt{Lithuania$set_region_codes()}
- \texttt{Lithuania$clean_common()}
- \texttt{Lithuania$clean_level_1()}
- \texttt{Lithuania$new()}
- \texttt{Lithuania$clone()}

Method \texttt{set_region_codes():} Set up a table of region codes for clean data

Usage:

\texttt{Lithuania$set_region_codes()}

Method \texttt{clean_common():} Common data cleaning for both levels

Usage:

\texttt{Lithuania$clean_common()}

Method \texttt{clean_level_1():} Lithuania Specific County Level Data Cleaning

Aggregates data to the level 1 (county) regional level. Data is provided by the source at the level 2 (municipality) regional level.
Usage:
Lithuania$clean_level_1()

Method new(): Initialize the country

Usage:
Lithuania$new(
  death_definition = "of",
  recovered_definition = "official",
  all_osp_fields = FALSE,
  national_data = FALSE,
  ...
)

Arguments:

death_definition A character string. Determines which criteria for attributing deaths to COVID is used. Should be "of", "with", or "after". Can also be "daily_deaths_def1", "daily_deaths_def2", or "daily_deaths_def3". (Defaults to "of", the strictest definition.)

recovered_definition A character string. Determines whether the count of officially-recovered (de jure) cases is used, or the statistical estimate provided by OSP. Should be "official" or "statistical". (Defaults to "official".)

all_osp_fields A logical scalar. Should all the meaningful data fields from the OSP source be returned? (Defaults FALSE)

national_data A logical scalar. Should national values be returned? (Defaults FALSE)

... Parameters passed to DataClass() initialize

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

Usage:
Lithuania$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Source

https://hub.arcgis.com/datasets/d49a63c934be4f65a93b6273785a8449_0

See Also

Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Mexico, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples

## Not run:
region <- Lithuania$new(verbos = TRUE, steps = TRUE, get = TRUE)

## End(Not run)
lithuania_codes  

Region Codes for Lithuania Dataset.

Description
The region codes for Lithuania

Usage
lithuania_codes

Format
An object of class spec_tbl_df (inherits from tbl_df, tbl, data.frame) with 61 rows and 6 columns.

Value
A tibble of region codes and related information, including ISO 3166:2 codes for counties (apskritis) and municipalities (savivaldybe), and noting which municipalities are city municipalities or regional municipalities.

make_github_workflow  

Create github action for a given source

Description
Makes a github workflow yaml file for a given source to be used as an action to check the data as a github action.

Usage
make_github_workflow(
  source,
  workflow_path = paste0(".github/workflows/", source, ".yaml"),
  cron = "36 12 * * *")

Arguments
source character_array The name of the class to create the workflow for.
workflow_path character_array The path to where the workflow file should be saved. Defaults to '.github/workflows/'
cron character_array the cron time to run the tests, defaults to 36 12 * * *, following the minute, hour, day(month), month and day(week) format.
**make_new_data_source**  
Create new country class for a given source

**Description**

Makes a new regional or national country class with the name provided as the source. This forms a basic template for the user to fill in with the specific field values and cleaning functions required. This also creates a github workflow file for the same country.

**Usage**

```r
make_new_data_source(
  source,
  type = "subnational",
  newfile_path = paste0("R/", source, ".R")
)
```

**Arguments**

- `source` character_array The name of the class to create. Must start with a capital letter (be upper camel case or an acronym in all caps such as WHO).
- `type` character_array the type of class to create, subnational or National defaults to subnational. Regional classes are individual countries, such as UK, Italy, India, etc. These inherit from `DataClass`, whilst national classes are sources for multiple countries data, such as JRC, JHU, Google, etc. These inherit from `CountryDataClass`.
- `newfile_path` character_array the place to save the class file

**message_verbose**  
Wrapper for message

**Description**

A wrapper for `message` that only prints output when `verbose = TRUE`.

**Usage**

```r
message_verbose(verbos = TRUE, ...)
```

**Arguments**

- `verbose` Logical, defaults to TRUE. Should verbose processing messages and warnings be returned.
- `...` Additional arguments passed to message.
Mexico

Mexico Class for downloading, cleaning and processing notification data

Description

Information for downloading, cleaning and processing COVID-19 region data for Mexico.

Notes on region codes:
Level 2 codes = INEGI Mexican official statistics geocoding, source: raw data
Level 1 INEGI codes are the first 2 characters of Level 2 INEGI codes

Super class

covidregionaldata::DataClass -> Mexico

Public fields

origin name of origin to fetch data for

supported_levels A list of supported levels.

supported_region_names A list of region names in order of level.

supported_region_codes A list of region codes in order of level.

common_data_urls List of named links to raw data.

level_data_urls List of named links to raw data that are level specific.

source_data_cols existing columns within the raw data

source_text Plain text description of the source of the data

source_url Website address for explanation/introduction of the data

Methods

Public methods:

- Mexico$set_region_codes()
- Mexico$download()
- Mexico$clean_common()
- Mexico$clean_level_1()
- Mexico$clean_level_2()
- Mexico$clone()

Method set_region_codes(): Set up a table of region codes for clean data

Usage:

Mexico$set_region_codes()
Method download(): Data download() function for Mexico data. This replaces the generic download function in DataClass(). To get the latest data use a PHP script from the website.

Usage:
Mexico$download()

Method clean_common(): Common Data Cleaning

Usage:
Mexico$clean_common()

Method clean_level_1(): Estados Level Data Cleaning

Usage:
Mexico$clean_level_1()

Method clean_level_2(): Municipality Level Data Cleaning

Usage:
Mexico$clean_level_2()

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

Usage:
Mexico$clone(deep = FALSE)

Arguments:
dee Whether to make a deep clone.

Source

https://datos.covid-19.conacyt.mx/

See Also

Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Netherlands, SouthAfrica, Switzerland, UK, USA

Examples

## Not run:
region <- Mexico$new(verbos TRUE, steps = TRUE, get = TRUE)
region$return()

## End(Not run)
mexico_codes  

*Region Codes for Mexico Dataset.*

**Description**

Details of the region codes used for the Mexico dataset.

**Usage**

mexico_codes

**Format**

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 2489 rows and 4 columns.

**Value**

A nested tibble of region codes and related information.

Netherlands  

*Netherlands Class for downloading, cleaning and processing notification data*

**Description**

Class for downloading, cleaning and processing COVID-19 sub-regional data for the Netherlands, provided by RVIM (English: National Institute for Public Health and the Environment). This data contains number of newly reported cases (that have tested positive), number of newly reported hospital admissions and number of newly reported deaths going back to 27/02/2020. Data is provided at both the province and municipality level.

**Super class**

`covidregionaldata::DataClass` -> Netherlands

**Public fields**

- `origin`  name of origin to fetch data for
- `supported_levels`  A list of supported levels.
- `supported_region_names`  A list of region names in order of level.
- `supported_region_codes`  A list of region codes in order of level.
- `common_data_urls`  List of named links to raw data. The first, and only entry, is be named main.
- `source_data_cols`  existing columns within the raw data
- `source_text`  Plain text description of the source of the data
- `source_url`  Website address for explanation/introduction of the data
Methods

Public methods:

• Netherlands$set_region_codes()
• Netherlands$clean_common()
• Netherlands$clean_level_1()
• Netherlands$clone()

Method set_region_codes(): Set up a table of region codes for clean data

Usage:
Netherlands$set_region_codes()

Method clean_common(): Common cleaning steps to be applied to raw data, regardless of level (province or municipality) for raw Netherlands data.

Usage:
Netherlands$clean_common()

Method clean_level_1(): Netherlands specific province level data cleaning. Takes the data cleaned by clean_common and aggregates it to the Province level (level 1).

Usage:
Netherlands$clean_level_1()

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

Usage:
Netherlands$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Source

https://data.rivm.nl/geonetwork/srv/dut/catalog.search#/metadata/5f6bc429-1596-490e-8618-1ed8fd7684e2?tab=relations

See Also

Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, SouthAfrica, Switzerland, UK, USA

Examples

## Not run:
region <- Netherlands$new(verse = TRUE, steps = TRUE, get = TRUE)
region$return()

## End(Not run)
process_internal  

*Internal Shared Regional Dataset Processing*

**Description**

Internal shared regional data cleaning designed to be called by `process`.

**Usage**

```r
process_internal(
  clean_data,
  level,
  group_vars,
  totals = FALSE,
  localise = TRUE,
  verbose = TRUE,
  process_fns
)
```

**Arguments**

- **clean_data**: The clean data for a class, e.g. `Italy$data$clean`
- **level**: The level of the data, e.g. 'level_1_region'
- **group_vars**: Grouping variables, used to for grouping and to localise names. It is assumed that the first entry indicates the main region variable and the second indicates the geocode for this variable.
- **totals**: Logical, defaults to `FALSE`. If `TRUE`, returns totalled data per region up to today's date. If `FALSE`, returns the full dataset stratified by date and region.
- **localise**: Logical, defaults to `TRUE`. Should region names be localised.
- **verbose**: Logical, defaults to `TRUE`. Should verbose processing messages and warnings be returned.
- **process_fns**: array, additional functions to be called after default processing steps

**See Also**

Functions used in the processing pipeline `run_default_processing_fns()`, `run_optional_processing_fns()`
region_dispatch

Control Grouping Variables used in process_internal

Description

Controls the grouping variables used in process_internal based on the supported regions present in the class.

Usage

region_dispatch(level, all_levels, region_names, region_codes)

Arguments

level A character string indicating the current level.
all_levels A character vector indicating all the levels supported.
region_names A named list of region names named after the levels supported.
region_codes A named list of region codes named after the levels supported.

reset_cache

Reset Cache and Update all Local Data

Description

Reset Cache and Update all Local Data

Usage

reset_cache()

Value

Null
return_data  

**Control data return**

**Description**

Controls data return for get_reigonal_data and get_national_data

**Usage**

```
return_data(obj, class = FALSE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>A Class based on a DataClass</td>
</tr>
<tr>
<td>class</td>
<td>Logical, defaults to FALSE. If TRUE returns the DataClass object rather than a tibble or a list of tibbles. Overrides steps.</td>
</tr>
</tbody>
</table>

run_default_processing_fns  

**Default processing steps to run**

**Description**

The default processing steps to which are always run. Runs on clean data

**Usage**

```
run_default_processing_fns(data)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A data table</td>
</tr>
</tbody>
</table>

**See Also**

Functions used in the processing pipeline `process_internal()`, `run_optional_processing_fns()`
run_optional_processing_fns

Optional processing steps to run

Description
user supplied processing steps which are run after default steps

Usage
run_optional_processing_fns(data, process_fns)

Arguments
- data: A data table
- process_fns: array, additional functions to be called after default processing steps

See Also
Functions used in the processing pipeline process_internal(), run_default_processing_fns()

set_negative_values_to_zero
Set negative data to 0

Description
Set data values to 0 if they are negative in a dataset. Data in the datasets should always be > 0.

Usage
set_negative_values_to_zero(data)

Arguments
- data: A data frame

Value
A data frame with all relevant data > 0.

See Also
Optional processing function totalise_data()
SouthAfrica

*SouthAfrica* Class for downloading, cleaning and processing notification data

**Description**

Information for downloading, cleaning and processing COVID-19 region data for South Africa.

**Super class**

covidregionaldata::DataClass -> SouthAfrica

**Public fields**

- origin  name of origin to fetch data for
- supported_levels  A list of supported levels.
- supported_region_names  A list of region names in order of level.
- supported_region_codes  A list of region codes in order of level.
- common_data_urls  List of named links to raw data.
- source_data_cols  existing columns within the raw data
- source_text  Plain text description of the source of the data
- source_url  Website address for explanation/introduction of the data

**Methods**

**Public methods:**

- *SouthAfrica*$set_region_codes()
- *SouthAfrica*$clean_common()
- *SouthAfrica*$clone()

**Method** `set_region_codes()`: Set up a table of region codes for clean data

*Usage:*

SouthAfrica$set_region_codes()

**Method** `clean_common()`: Province level data cleaning

*Usage:*

SouthAfrica$clean_common()

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

*Usage:*

SouthAfrica$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.
start_using_memoise

Source

https://github.com/dsfsi/covid19za/

See Also

Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, Switzerland, UK, USA

Examples

```r
## Not run:
region <- SouthAfrica$new(verbos = TRUE, steps = TRUE, get = TRUE)
region$return()
## End(Not run)
```

start_using_memoise  Add useMemoise to options

Description

Adds useMemoise to options meaning memoise is used when reading data in.

Usage

```r
start_using_memoise(path = tempdir(), verbose = TRUE)
```

Arguments

- `path`  Path to cache directory, defaults to a temporary directory.
- `verbose`  Logical, defaults to TRUE. Should verbose processing messages and warnings be returned.

stop_using_memoise  Stop using useMemoise

Description

Sets useMemoise in options to NULL, meaning memoise isn’t used when reading data in

Usage

```r
stop_using_memoise()
```
Switzerland

Switzerland Class for downloading, cleaning and processing notification data

Description

Information for downloading, cleaning and processing COVID-19 region data for Switzerland

Liechtenstein

Liechtenstein is not a canton of Switzerland, but is presented in the source data as a peer of Swiss cantons and assigned the two letter code FL. covidregionaldata modifies this and presents the region code for Liechtenstein as FL-FL, consistent with the Swiss ISO 3166-2 codes which are of the form CH-BE, CH-ZH, CH-VD, ...

If you do not wish to work with Liechtenstein data, filter out on this code. Note that this is labelled as a ISO 3166-2 code but Liechtenstein’s real ISO 3166-2 codes refer to sub-national regions.

Additional data

In addition to the standard covidregionaldata columns provided, the OpenDataZH source data provides other figures for ICU occupancy, number of patients on ventilators, and the how many individuals are isolated or quarantined. These columns are passed through unchanged. Further detail on them can be found at https://github.com/openZH/covid_19/#swiss-cantons-and-principality-of-liechtenstein

Super class

covidregionaldata::DataClass -> Switzerland

Public fields

origin name of origin to fetch data for
supported_levels A list of supported levels.
supported_region_names A list of region names in order of level.
supported_region_codes A list of region codes in order of level.
common_data_urls List of named links to raw data.
source_data_cols existing columns within the raw data
source_text Plain text description of the source of the data
source_url Website address for explanation/introduction of the data

Methods

Public methods:

• Switzerland$set_region_codes()
• Switzerland$clean_common()
• Switzerland$clone()
Method set_region_codes(): Set up a table of region codes for clean data

Usage:
Switzerland$set_region_codes()

Method clean_common(): Switzerland specific state level data cleaning

Usage:
Switzerland$clean_common()

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

Usage:
Switzerland$clone(deep = FALSE)

Arguments:
  deep  Whether to make a deep clone.

Source
https://github.com/openZH/covid_19/

See Also
Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, UK, USA

Examples

```r
## Not run:
region <- Switzerland$new(verbose = TRUE, steps = TRUE, get = TRUE)
region$return()
## End(Not run)
```

test_cleaning  Test clean method works correctly

Description
Test data can be cleaned properly. The clean method is invoked to generate clean data. This data is checked to ensure it is a data.frame, is not empty, has at least two columns and that columns are clean by calling expect_clean_cols. Also tests that available_regions() are not NA and they are all characters.

Usage
test_cleaning(DataClass_obj)
Arguments

DataClass_obj  The R6Class object to perform checks on. Must be a DataClass or DataClass child object.

See Also

Functions used for testing data is cleaned and processed correctly `expect_clean_cols()`., `expect_columns_contain_data()`, `expect_processed_cols()`, `test_download()`, `test_processing()`, `test_return()`

test_download  Test download method works correctly

Description

Test data can be downloaded if `download = TRUE`, or a requested snapshot file is not found, and store a snapshot in the `snapshot_dir`. If an existing snapshot file is found then load this data to use in future tests.

Usage

test_download(DataClass_obj, download, snapshot_path)

Arguments

DataClass_obj  The R6Class object to perform checks on. Must be a DataClass or DataClass child object.
download Logical check to download or use a snapshot of the data
snapshot_path  character_array the path to save the downloaded snapshot to.

See Also

Functions used for testing data is cleaned and processed correctly `expect_clean_cols()`., `expect_columns_contain_data()`, `expect_processed_cols()`, `test_cleaning()`, `test_processing()`, `test_return()`

test_processing  Test process method works correctly

Description

Test data can be processed correctly using the process method. process is invoked to generate processed data which is then checked to ensure it is a data.frame, which is not empty, has at least 2 columns and calls `expect_processed_columns` to check each column types.

Usage

test_processing(DataClass_obj, all = FALSE)
**Arguments**

- **DataClass_obj**  The R6Class object to perform checks on. Must be a DataClass or DataClass child object.
- **all** Logical. Run tests with all settings (TRUE) or with those defined in the current class instance (FALSE). Defaults to FALSE.

**See Also**

Functions used for testing data is cleaned and processed correctly: `expect_clean_cols()`, `expect_columns_contain_data()`, `expect_processed_cols()`, `test_cleaning()`, `test_download()`, `test_return()`

---

**Description**

Test data can be returned correctly using the return method. `return` is invoked to generate returned data which is then checked to ensure it is a data.frame, not empty and has at least 2 columns. Each column is then checked to ensure it contains data and is not just composed of NAs.

**Usage**

```r
test_return(DataClass_obj)
```

**Arguments**

- **DataClass_obj**  The R6Class object to perform checks on. Must be a DataClass or DataClass child object.

**See Also**

Functions used for testing data is cleaned and processed correctly: `expect_clean_cols()`, `expect_columns_contain_data()`, `expect_processed_cols()`, `test_cleaning()`, `test_download()`, `test_processing()`

---

**totalise_data**  

Get totals data given the time series data.

**Description**

Get totals data given the time series data.

**Usage**

```r
totalise_data(data)
```
Arguments

data A data table

Value

A data table, totalled up

See Also

Optional processing function `set_negative_values_to_zero()`

---

UK

*United Kingdom Class for downloading, cleaning and processing notification data.*

Description

Extracts daily COVID-19 data for the UK, stratified by region and nation. Additional options for this class are: to return subnational English regions using NHS region boundaries instead of PHE boundaries (`nhsregions = TRUE`), a release date to download from (`release_date`) and a geographical resolution (`resolution`).

Super class

`covidregionaldata::DataClass` -> UK

Public fields

- `origin` name of origin to fetch data for
- `supported_levels` A list of supported levels.
- `supported_region_names` A list of region names in order of level.
- `supported_region_codes` A list of region codes in order of level.
- `common_data_urls` List of named links to raw data. The first, and only entry, is be named main.
- `level_data_urls` List of named links to raw data that are level specific.
- `source_data_cols` existing columns within the raw data
- `source_text` Plain text description of the source of the data
- `source_url` Website address for explanation/introduction of the data
- `query_filters` Set what filters to use to query the data
- `nhsregions` Whether to include NHS regions in the data
- `release_date` The release date for the data
- `resolution` The resolution of the data to return
- `authority_data` The raw data for creating authority lookup tables
Methods

**Public methods:**

- `UK$set_region_codes()`
- `UK$download()`
- `UK$clean_level_1()`
- `UK$clean_level_2()`
- `UK$new()`
- `UK$download_filter()`
- `UK$set_filters()`
- `UK$download_nhs_regions()`
- `UK$add_nhs_regions()`
- `UK$specific_tests()`
- `UK$clone()`

**Method** `set_region_codes()`: Specific function for getting region codes for UK.

*Usage:*

`UK$set_region_codes()`

**Method** `download()`: UK specific download function.

*Usage:*

`UK$download()`

**Method** `clean_level_1()`: Region Level Data Cleaning

*Usage:*

`UK$clean_level_1()`

**Method** `clean_level_2()`: Level 2 Data Cleaning

*Usage:*

`UK$clean_level_2()`

**Method** `new()`: Initialize the UK Class

*Usage:*

`UK$new(nhsregions = FALSE, release_date = NULL, resolution = "utla", ...)`

*Arguments:*

- `nhsregions` Return subnational English regions using NHS region boundaries instead of PHE boundaries.
- `release_date` Date data was released. Default is to extract latest release. Dates should be in the format "yyyy-mm-dd".
- `resolution` "utla" (default) or "ltla", depending on which geographical resolution is preferred ...

*Optional arguments passed to DataClass() initialize.*

*Examples:*
Method download_filter(): Helper function for downloading data API

Usage:
UK$download_filter(filter)

Arguments:
filter region filters

Method set_filters(): Set filters for UK data api query.

Usage:
UK$set_filters()

Method download_nhs_regions(): Download NHS data for level 1 regions Separate NHS data is available for "first" admissions, excluding readmissions. This is available for England + English regions only. Data are available separately for the periods 2020-08-01 to 2021-04-06, and 2021-04-07 - present. See: https://www.england.nhs.uk/statistics/statistical-work-areas/covid-19-hospital-activity/ Section 2, "2. Estimated new hospital cases"

Usage:
UK$download_nhs_regions()

Returns: nhs data.frame of nhs regions

Method add_nhs_regions(): Add NHS data for level 1 regions Separate NHS data is available for "first" admissions, excluding readmissions. This is available for England + English regions only. See: https://www.england.nhs.uk/statistics/statistical-work-areas/covid-19-hospital-activity/ Section 2, "2. Estimated new hospital cases"

Usage:
UK$add_nhs_regions(clean_data, nhs_data)

Arguments:
clean_data Cleaned UK covid-19 data
nhs_data NHS region data

Method specific_tests(): Specific tests for UK data. In addition to generic tests ran by DataClass$test() data for NHS regions are downloaded and ran through the same generic checks (test_cleaning, test_processing, test_return). If download = TRUE or a snapshot file is not found, the nhs data is downloaded and saved to the snapshot location provided. If an existing snapshot file is found then this data is used in the next tests. Tests data can be downloaded, cleaned, processed and returned. Designed to be ran from test and not ran directly.

Usage:
UK$specific_tests(
  self_copy,
  download = FALSE,
  all = FALSE,
  snapshot_path = "",
  ...
)

Arguments:
self_copy R6class the object to test.
download logical. To download the data (TRUE) or use a snapshot (FALSE). Defaults to FALSE.
all logical. Run tests with all settings (TRUE) or with those defined in the current class instance (FALSE). Defaults to FALSE.
snapshot_path character_array the path to save the downloaded snapshot to. Works on the snapshot path constructed by test but adds
... Additional parameters to pass to specific_tests

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

Usage:
UK$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Source

https://coronavirus.data.gov.uk/details/download
https://coronavirus.data.gov.uk/details/download

See Also

Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, USA

Examples

## Not run:
# setup a data cache
start_using_memoise()

# download, clean and process level 1 UK data with hospital admissions
region <- UK$new(level = "1", nhsregions = TRUE)
region$return()

# initialise level 2 data
utla <- UK$new(level = "2")

# download UTLA data
uk_codes

uk_codes

Region Codes for UK Dataset.

Description

The uk authority look table for providing region codes used for level 2 UK data.

Usage

uk_codes

Format

An object of class tbl_df (inherits from tbl, data.frame) with 429 rows and 4 columns.
USA

Value
A tibble of region codes and related information.

---

USA Class for downloading, cleaning and processing notification data

Description
Information for downloading, cleaning and processing COVID-19 region data for USA.

Super class

covidregionaldata::DataClass -&gt; USA

Public fields

origin  name of origin to fetch data for
supported_levels  A list of supported levels.
supported_region_names  A list of region names in order of level.
supported_region_codes  A list of region codes in order of level.
level_data_urls  List of named links to raw data that are level specific.
source_data_cols  existing columns within the raw data
source_text  Plain text description of the source of the data
source_url  Website address for explanation/introduction of the data

Methods

Public methods:

• USA$set_region_codes()
• USA$clean_level_1()
• USA$clean_level_2()
• USA$clone()

Method set_region_codes(): Set up a table of region codes for clean data
Usage:
USA$set_region_codes()

Method clean_level_1(): State Level Data Cleaning
Usage:
USA$clean_level_1()

Method clean_level_2(): County Level Data Cleaning
Usage:
WHO$clean_level_2()

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

**Usage:**
USA$clone(deep = FALSE)

**Arguments:**
derep Whether to make a deep clone.

**Source**
https://github.com/nytimes/covid-19-data/

**See Also**
Subnational data sources Belgium, Brazil, Canada, Colombia, Covid19DataHub, Cuba, France, Germany, Google, India, Italy, JHU, Lithuania, Mexico, Netherlands, SouthAfrica, Switzerland, UK

**Examples**
```r
## Not run:
region <- USA$new(verbse = TRUE, steps = TRUE, get = TRUE)
region$return()
## End(Not run)
```

---

**WHO**

**R6 Class containing specific attributes and methods for World Health Organisation data**

**Description**
Information for downloading, cleaning and processing COVID-19 region data from the World Health Organisation

**Super classes**

covidregionaldata::DataClass <- covidregionaldata::CountryDataClass -> WHO

**Public fields**

- **origin** name of origin to fetch data for
- **supported_levels** A list of supported levels.
- **supported_region_names** A list of region names in order of level.
- **supported_region_codes** A list of region codes in order of level.
- **common_data_urls** List of named links to raw data. The first, and only entry, is be named main.
- **source_data_cols** existing columns within the raw data
- **source_text** Plain text description of the source of the data
- **source_url** Website address for explanation/introduction of the data
WHO

Methods

Public methods:

- WHO$clean_common()
- WHO$return()
- WHO$specific_tests()
- WHO$clone()

**Method clean_common()**: WHO specific data cleaning

*Usage:*

WHO$clean_common()

**Method return()**: Specific return settings for the WHO dataset.

*Usage:*

WHO$return()

**Method specific_tests()**: Run additional tests on WHO data. Tests that there is only one row per country. Designed to be ran from test and not ran directly.

*Usage:*

WHO$specific_tests(self_copy, ...)

*Arguments:*

- self_copy R6class the object to test
- ... Extra params passed to specific download functions

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

*Usage:*

WHO$clone(deep = FALSE)

*Arguments:*

- deep Whether to make a deep clone.

Source

https://covid19.who.int/

See Also

National data sources Covid19DataHub, ECDC, Google, JHU, JRC

Examples

```r
## Not run:
national <- WHO$new(verbose = TRUE, steps = TRUE, get = TRUE)
national$return()

## End(Not run)
```
Index

* **aggregations**
  - Covid19DataHub, 14
  - Google, 36
  - JHU, 43

* **compulsory_processing**
  - add_extra_na_cols, 4
  - calculate_columns_from_existing_data, 8
  - complete_cumulative_columns, 12
  - fill_empty_dates_with_na, 26

* **datasets**
  - all_country_data, 4
  - colombia_codes, 12
  - france_codes, 28
  - JHU_codes, 45
  - lithuania_codes, 52
  - mexico_codes, 56
  - uk_codes, 72

* **dataset**
  - Belgium, 5
  - Brazil, 6
  - Canada, 9
  - Colombia, 10
  - Covid19DataHub, 14
  - Cuba, 16
  - ECDC, 23
  - France, 27
  - Germany, 29
  - Google, 36
  - India, 38
  - Italy, 42
  - JHU, 43
  - JRC, 46
  - Lithuania, 48
  - Mexico, 54
  - Netherlands, 56
  - SouthAfrica, 62
  - Switzerland, 64
  - UK, 68
  - USA, 73
  - WHO, 74

* **interface**
  - CountryDataClass, 13
  - DataClass, 18
  - get_available_datasets, 30
  - get_national_data, 31
  - get_regional_data, 33
  - initialise_dataclass, 40

* **national**
  - Covid19DataHub, 14
  - ECDC, 23
  - Google, 36
  - JHU, 43
  - JRC, 46
  - WHO, 74

* **optional_processing**
  - set_negative_values_to_zero, 61
  - totalise_data, 67

* **processing**
  - process_internal, 58
  - run_default_processing_fns, 60
  - run_optional_processing_fns, 61

* **subnational**
  - Belgium, 5
  - Brazil, 6
  - Canada, 9
  - Colombia, 10
  - Covid19DataHub, 14
  - Cuba, 16
  - France, 27
  - Germany, 29
  - Google, 36
  - India, 38
  - Italy, 42
  - JHU, 43
  - JRC, 46
  - Lithuania, 48
  - Mexico, 54
  - Netherlands, 56
SouthAfrica, 62
Switzerland, 64
UK, 68
USA, 73

* tests
  expect_clean_cols, 25
  expect_columns_contain_data, 25
  expect_processed_cols, 26
  test_cleaning, 65
  test_download, 66
  test_processing, 66
  test_return, 67

* utility
  all_country_data, 4
  check_level, 10
  csv_reader, 16
  download_excel, 23
  glue_level, 35
  make_github_workflow, 52
  make_new_data_source, 53
  message_verbose, 53
  process_internal, 58
  region_dispatch, 59
  reset_cache, 59
  return_data, 60
  run_default_processing_fns, 60
  run_optional_processing_fns, 61
  start_using_memoise, 63
  stop_using_memoise, 63

  add_extra_na_cols, 4, 8, 12, 27
  all_country_data, 4

Belgium, 5, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74
Brazil, 6, 6, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74

calculate_columns_from_existing_data, 4, 8, 12, 27
Canada, 6, 8, 9, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74
check_level, 10
Colombia, 6, 8, 10, 10, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74
colombia_codes, 12
complete_cumulative_columns, 4, 8, 12, 27
CountryDataClass, 13, 22, 31, 32, 34, 41

Covid19DataHub, 6, 8, 10, 11, 14, 17, 25, 28, 30, 37, 39, 43, 45, 47, 51, 55, 57, 63, 65, 71, 74

Covidregionaldata::CountryDataClass, 14, 23, 36, 44, 46, 74
Covidregionaldata::DataClass, 5, 6, 9, 10, 13, 14, 16, 23, 27, 29, 36, 38, 42, 44, 46, 50, 54, 56, 62, 64, 68, 73, 74
csv_reader, 16
Cuba, 6, 8, 10, 11, 15, 16, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74

DataClass, 13, 18, 31, 32, 34, 41
DataClass(), 13, 51, 55, 69
download_excel, 23

ECDC, 15, 23, 37, 45, 47, 75
ECDC(), 32
expect_clean_cols, 25, 26, 66, 67
expect_columns_contain_data, 25, 25, 26, 66, 67
expect_processed_cols, 25, 26, 26, 66, 67
fill_empty_dates_with_na, 4, 8, 12, 26
France, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74
france_codes, 28

Germany, 6, 8, 10, 11, 15, 17, 28, 29, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74
get_available_datasets, 13, 22, 30, 32, 34, 41
get_national_data, 13, 22, 31, 32, 34, 41
get_regional_data, 13, 22, 31, 32, 33, 41
glue_level, 35
Google, 6, 8, 10, 11, 15, 17, 25, 28, 30, 36, 39, 43, 45, 47, 51, 55, 57, 63, 65, 71, 74, 75
Google(), 32

India, 6, 8, 10, 11, 15, 17, 28, 30, 37, 38, 43, 45, 51, 55, 57, 63, 65, 71, 74
initialise_dataclass, 13, 22, 31, 32, 34, 40
Italy, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 42, 45, 51, 55, 57, 63, 65, 71, 74
Italy(), 34

JHU, 6, 8, 10, 11, 15, 17, 25, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 74, 75
JHU(), 32
JHU_codes, 45
JRC, 15, 25, 37, 45, 46, 75

Lithuania, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 48, 55, 57, 63, 65, 71, 74
lithuania_codes, 52

make_github_workflow, 52
make_new_data_source, 53
message_verbose, 53

Mexico, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 54, 57, 63, 65, 71, 74
mexico_codes, 56

Netherlands, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 56, 63, 65, 71, 74

process_internal, 58, 60, 61
region_dispatch, 59
reset_cache, 59
return_data, 60
run_default_processing_fns, 58, 60, 61
run_optional_processing_fns, 58, 60, 61

set_negative_values_to_zero, 61, 68
SouthAfrica, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 62, 65, 71, 74
start_using_memoise, 63
stop_using_memoise, 63
Switzerland, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 64, 71, 74

test_cleaning, 25, 26, 65, 66, 67
test_download, 25, 26, 66, 66, 67
test_processing, 25, 26, 66, 66, 67
test_return, 25, 26, 66, 67, 67
totalise_data, 61, 67

UK, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 68, 74
UK(), 34
uk_codes, 72
USA, 6, 8, 10, 11, 15, 17, 28, 30, 37, 39, 43, 45, 51, 55, 57, 63, 65, 71, 73

WHO, 15, 25, 37, 45, 47, 74
WHO(), 32