Package ‘quincunx’

July 22, 2025

Type Package

Title REST API Client for the 'PGS' Catalog

Version 0.1.10

Description Programmatic access to the 'PGS' Catalog. This package provides easy access to 'PGS' Catalog data by accessing the REST API <`https://www.pgscatalog.org/rest/`>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Suggests testthat, knitr, rmarkdown, ggplot2

Imports stringr, vroom, purrr, glue, dplyr, tidyjson, tibble, lubridate, rlang, tidyr, httr, utils, rvest, progress, methods, writexl, memoise, readr

Collate 'ancestry_categories.R' 'cc.R' 'class-cohorts.R' 'class-performance_metrics.R' 'class-publications.R' 'class-releases.R' 'class-sample_sets.R' 'class-scores.R' 'class-trait_categories.R' 'class-traits.R' 'clear_cache.R' 'contains_question_mark.R' 'count.R' 'delay.R' 'drop_metadata_cols.R' 'first_non_na.R' 'generics.R' 'get.R' 'get_ancestry_categories.R' 'get_cohorts.R' 'get_column.R' 'get_performance_metrics.R' 'get_publications.R' 'get_releases.R' 'get_sample_sets.R' 'get_scores.R' 'get_trait_categories.R' 'get_traits.R' 'id_mapping.R' 'is_json_empty.R' 'is_paginated.R' 'is_pgs_id.R' 'is_pubmed_id.R' 'messages.R' 'n_pages.R' 'nr_to_na.R' 'offsets.R' 'open_in_dbsnp.R' 'open_in_pgs_catalog.R' 'open_in_pubmed.R' 'parse-ancestry_categories.R' 'parse-cohorts.R' 'parse-performance_metrics.R' 'parse-publications.R' 'parse-releases.R' 'parse-sample_sets.R' 'parse-scores.R' 'parse-trait_categories.R' 'parse-traits.R' 'parse_estimate.R' 'read_file_column_names.R' 'read_pgs_scoring_file.R' 'relocate_metadata_cols.R'

'remap_id.R' 'request.R' 's4-utils.R' 'stages.R' 'sure.R'
'unwrap_cohort.R' 'unwrap_demographics.R' 'unwrap_efotrait.R'
'unwrap_interval.R' 'unwrap_publication.R' 'unwrap_sample.R'
'warnings.R' 'write_xlsx.R'

Depends R (>= 4.1.0)

URL https://github.com/ramiromagno/quincunx,
https://rmagno.eu/quincunx/

BugReports https://github.com/ramiromagno/quincunx/issues

Config/Needs/website patterninstitute/chic

NeedsCompilation no

Author Ramiro Magno [aut, cre] (ORCID:
<https://orcid.org/0000-0001-5226-3441>),
Isabel Duarte [aut] (ORCID: <https://orcid.org/0000-0003-0060-2936>),
Ana-Teresa Maia [aut] (ORCID: <https://orcid.org/0000-0002-0454-9207>),
CINTESIS [cph, fnd],
Pattern Institute [cph, fnd] (ROR: <https://ror.org/04jrgd746>)

Maintainer Ramiro Magno <rmagno@pattern.institute>

Repository CRAN

Date/Publication 2025-05-31 17:10:02 UTC

Contents

`ancestry_categories` *Ancestry categories and classes*

Description

A dataset containing the ancestry categories defined in NHGRI-EBI GWAS Catalog framework (Table 1, doi:10.1186/s1305901813962). Ancestry categories are assigned to samples with distinct and well-defined patterns of genetic variation. You will find these categories in the variable `ancestry_category` of the following objects: scores, performance_metrics and sample_sets. Ancestry categories (`ancestry_category`) are further clustered into ancestry classes (`ancestry_class`).

Usage

```
ancestry_categories
```

Format

A data frame with 19 ancestry categories (rows) and 6 columns:

ancestry_category Ancestry category.

ancestry_class To reduce the complexity associated with the many ancestry categories, some have been merged into higher-level groupings (`ancestry_class`). These groupings represent the current breadth of data in the PGS Catalog and are likely to change as more data is added.

ancestry_class_symbol 3-letter code for the `ancestry_class` e.g. `"EUR"` or `"MAE"`.

ancestry_class_colour Hexadecimal colour code associated with ancestry groupings (`ancestry_class`). This can be useful when visually communicating about ancestries.

definition Description of the ancestry category.

examples Examples of detailed descriptions of sample ancestries included in the category.

Source

Table 1 of Moralles et al. (2018): doi:10.1186/s1305901813962

PGS Catalog Ancestry Documentation: `http://www.pgscatalog.org/docs/ancestry/`

Examples

```
ancestry_categories
```

`bind` *Bind PGS Catalog objects*

Description

Binds together PGS Catalog objects of the same class. Note that `bind()` preserves duplicates whereas `union` does not.

Usage

```
bind(x, ...)
```

Arguments

`x`	An object of either class scores, publications, traits, performance_metrics, sample_sets, cohorts or trait_categories.
`...`	Objects of the same class as `x`.

Value

An object of the same class as `x`.

Examples

```
# Get some `scores` objects:
my_scores_1 <- get_scores(c('PGS000012', 'PGS000013'))
my_scores_2 <- get_scores(c('PGS000013', 'PGS000014'))

# NB: with `bind()`, PGS000013 is repeated (as opposed to `union()`)
bind(my_scores_1, my_scores_2)@scores
```

`clear_cache` *Clear quincunx cache of memoised functions*

Description

quincunx uses memoised functions for the REST API calls. Use this function to reset the cache.

Usage

```
clear_cache()
```

Value

Returns a logical value, indicating whether the resetting of the cache was successful (`TRUE`) or not `FALSE`.

Examples

```
clear_cache()
```

`cohorts-class` *An S4 class to represent a set of cohorts*

Description

The cohorts object consists of two tables (slots) that combined form a relational database of a subset of cohorts. Each cohort is an observation (row) in the `cohorts` table (first table).

Slots

`cohorts` A table of cohorts. Each cohort (row) is identified by its `cohort_symbol`. Columns:

- **cohort_symbol** Cohort symbol. Example: `"CECILE"`.
- **cohort_name** Cohort full name. Example: `"CECILE Breast Cancer Study"`.

`pgs_ids` A table of cohorts and their associated polygenic scores identifiers. Columns:

- **cohort_symbol** Cohort symbol. Example: `"CECILE"`.
- **pgs_id** Polygenic Score (PGS) identifier.
- **stage** Sample stage: either `"gwas/dev"` or `"eval"`.

`get_ancestry_categories` *Get ancestry categories and classes*

Description

Retrieves ancestry categories and classes. This function simply returns the object `ancestry_categories`.

Usage

```
get_ancestry_categories()
```

Value

A tibble with ancestry categories, classes and associated information. See `ancestry_categories` for details about each column.

Examples

```
get_ancestry_categories()
```

`get_cohorts` *Get PGS Catalog Cohorts*

Description

Retrieves cohorts via the PGS Catalog REST API. Please note that all `cohort_symbol` is vectorised, thus allowing for batch mode search.

Usage

```
get_cohorts(
  cohort_symbol = NULL,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`cohort_symbol`	A cohort symbol or `NULL` if all cohorts are intended.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar indicating download progress from the REST API server.

Value

A cohorts object.

Examples

```
# Get information about specific cohorts by their symbols (acronyms)
get_cohorts(cohort_symbol = c('23andMe', 'IPOBCS'))

# Get info on all cohorts (may take a few minutes to download)
## Not run:
get_cohorts()

## End(Not run)
```

`get_performance_metrics`

Get PGS Catalog Performance Metrics

Description

Retrieves performance metrics via the PGS Catalog REST API. The REST API is queried multiple times with the criteria passed as arguments (see below). By default all performance metrics that match the criteria supplied in the arguments are retrieved: this corresponds to the default option `set_operation` set to `'union'`. If you rather have only the associations that match simultaneously all criteria provided, then set `set_operation` to `'intersection'`.

Usage

```
get_performance_metrics(
  ppm_id = NULL,
  pgs_id = NULL,
  set_operation = "union",
  interactive = TRUE,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`ppm_id`	A character vector of PGS Catalog performance metrics accession identifiers.
`pgs_id`	A `character` vector of PGS Catalog score accession identifiers.
`set_operation`	Either `'union'` or `'intersection'`. This tells how performance metrics retrieved by different criteria should be combined: `'union'` binds together all results removing duplicates and `'intersection'` only keeps same performance metrics found with different criteria.

`interactive`	A logical. If all performance metrics are requested, whether to ask interactively if we really want to proceed.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Details

Please note that all search criteria are vectorised, thus allowing for batch mode search.

Value

A performance_metrics object.

Examples

```
## Not run:
# Get performance metrics catalogued with identifier 'PPM000001'
get_performance_metrics(ppm_id = 'PPM000001')

# Get performance metrics associated with polygenic score id 'PGS000001'
get_performance_metrics(pgs_id = 'PGS000001')

# To retrieve all catalogued performed metrics in PGS Catalog you simply
# leave the parameters `ppm_id` and `pgs_id` as `NULL`.
get_performance_metrics()

## End(Not run)
```

`get_publications` *Get PGS Catalog Publications*

Description

Retrieves PGS publications via the PGS Catalog REST API. The REST API is queried multiple times with the criteria passed as arguments (see below). By default all publications that match the criteria supplied in the arguments are retrieved: this corresponds to the default option `set_operation` set to `'union'`. If you rather have only the associations that match simultaneously all criteria provided, then set `set_operation` to `'intersection'`.

Usage

```
get_publications(
  pgp_id = NULL,
  pgs_id = NULL,
  pubmed_id = NULL,
  author = NULL,
  set_operation = "union",
  interactive = TRUE,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgp_id`	A character vector of PGS Catalog publication accession identifiers.
`pgs_id`	A `character` vector of PGS Catalog score accession identifiers.
`pubmed_id`	An `integer` vector of PubMed identifiers.
`author`	A character vector of author names, any author in the list of authors in a publication, .e.g. `'Mavaddat'`.
`set_operation`	Either `'union'` or `'intersection'`. This tells how publications retrieved by different criteria should be combined: `'union'` binds together all results removing duplicates and `'intersection'` only keeps same publications found with different criteria.
`interactive`	A logical. If all publications are requested, whether to ask interactively if we really want to proceed.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Details

Please note that all search criteria are vectorised, thus allowing for batch mode search. For more details see the help vignette: `vignette("getting-pgs-publications", package = "quincunx")`.

Value

A publications object.

Examples

```
## Not run:
# Get PGS publications by their identifier
get_publications(pgp_id = c('PGP000001', 'PGP000002'))

# By polygenic score identifier
```

```
get_publications(pgs_id = 'PGS000003')

# By PubMed identifier
get_publications(pubmed_id = '30554720')

# By author's last name
get_publications(author = 'Natarajan')

## End(Not run)
```

`get_releases` *Get PGS Catalog Releases*

Description

This function retrieves PGS Catalog release information. Note that the columns `pgs_id`, `ppm_id` and `pgp_id` contain in each element a vector. These columns can be unnested using `unnest_longer` (see Examples).

Usage

```
get_releases(
  date = "latest",
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`date`	One or more dates formatted as `"YYYY-MM-DD"` for respective releases, `"latest"` for the latest release, or `"all"` for all releases.
`verbose`	Whether to print information about the underlying requests to the REST API server.
`warnings`	Whether to print warnings about the underlying requests to the REST API server.
`progress_bar`	Whether to show a progress bar indicating download progress from the REST API server.

Value

A data frame where each row is a release. Columns are:

date Release date.

n_pgs Number of released Polygenic Score (PGS) identifiers (`pgs_id`).

n_ppm Number of released Performance Metric (PPM) identifiers (`ppm_id`).

n_pgp Number of released PGS Catalog Publication (PGP) identifiers (`pgp_id`).

pgs_id Released Polygenic Score (PGS) identifiers.

ppm_id Released Performance Metric (PPM) identifiers.

pgp_id Released PGS Catalog Publication (PGP) identifiers.

notes News about the release.

Examples

```
## Not run:
# Get the latest release
get_releases()
get_releases(date = 'latest')

# Get all releases
get_releases(date = 'all')

# Get a specific release by date
get_releases(date = '2020-08-19')

## End(Not run)
```

`get_sample_sets` *Get PGS Catalog Sample Sets*

Description

Retrieves sample sets via the PGS Catalog REST API. The REST API is queried multiple times with the criteria passed as arguments (see below). By default all sample sets that match the criteria supplied in the arguments are retrieved: this corresponds to the default option `set_operation` set to `'union'`. If you rather have only the associations that match simultaneously all criteria provided, then set `set_operation` to `'intersection'`.

Usage

```
get_sample_sets(
  pss_id = NULL,
  pgs_id = NULL,
  set_operation = "union",
  interactive = TRUE,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pss_id`	A character vector of PGS Catalog sample sets accession identifiers.
`pgs_id`	A `character` vector of PGS Catalog score accession identifiers.
`set_operation`	Either `'union'` or `'intersection'`. This tells how performance metrics retrieved by different criteria should be combined: `'union'` binds together all results removing duplicates and `'intersection'` only keeps same sample sets found with different criteria.
`interactive`	A logical. If all sample sets are requested, whether to ask interactively if we really want to proceed.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar indicating download progress from the REST API server.

Details

Please note that all search criteria are vectorised, thus allowing for batch mode search.

Value

A sample_sets object.

Examples

```
## Not run:
# Search by PGS identifier
get_sample_sets(pgs_id = 'PGS000013')

# Search by the PSS identifier
get_sample_sets(pss_id = 'PSS000068')

## End(Not run)
```

`get_scores` *Get PGS Catalog Scores*

Description

Retrieves polygenic scores via the PGS Catalog REST API. The REST API is queried multiple times with the criteria passed as arguments (see below). By default all scores that match the criteria supplied in the arguments are retrieved: this corresponds to the default option `set_operation` set to `'union'`. If you rather have only the associations that match simultaneously all criteria provided, then set `set_operation` to `'intersection'`.

Usage

```
get_scores(
  pgs_id = NULL,
  efo_id = NULL,
  pubmed_id = NULL,
  set_operation = "union",
  interactive = TRUE,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgs_id`	A `character` vector of PGS Catalog score accession identifiers.
`efo_id`	A character vector of EFO identifiers.
`pubmed_id`	An `integer` vector of PubMed identifiers.
`set_operation`	Either `'union'` or `'intersection'`. This tells how scores retrieved by different criteria should be combined: `'union'` binds together all results removing duplicates and `'intersection'` only keeps same scores found with different criteria.
`interactive`	A logical. If all scores are requested, whether to ask interactively if we really want to proceed.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Details

Please note that all search criteria are vectorised, thus allowing for batch mode search.

Value

A scores object.

Examples

```
## Not run:
# By `pgs_id`
get_scores(pgs_id = 'PGS000088')

# By `efo_id`
get_scores(efo_id = 'EFO_0007992')

# By `pubmed_id`
get_scores(pubmed_id = '25748612')
```

```
## End(Not run)
```

`get_traits` *Get PGS Catalog Traits*

Description

Retrieves traits via the PGS Catalog REST API. The REST API is queried multiple times with the criteria passed as arguments (see below). By default all traits that match the criteria supplied in the arguments are retrieved: this corresponds to the default option `set_operation` set to `'union'`. If you rather have only the traits that match simultaneously all criteria provided, then set `set_operation` to `'intersection'`.

Usage

```
get_traits(
  efo_id = NULL,
  trait_term = NULL,
  exact_term = TRUE,
  include_children = FALSE,
  set_operation = "union",
  interactive = TRUE,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`efo_id`	A character vector of EFO identifiers.
`trait_term`	A character vector of terms to be matched against trait identifiers (`efo_id`), trait descriptions, synonyms thereof, externally mapped terms, or even trait categories.
`exact_term`	A logical value, indicating whether to match the `trait_term` exactly (`TRUE`) or not (`FALSE`).
`include_children`	A logical value, indicating whether to include child traits or not.
`set_operation`	Either `'union'` or `'intersection'`. This tells how performance metrics retrieved by different criteria should be combined: `'union'` binds together all results removing duplicates and `'intersection'` only keeps same sample sets found with different criteria.
`interactive`	A logical. If all traits are requested, whether to ask interactively if we really want to proceed.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.

`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar indicating download progress from the REST API server.

Details

Please note that all search criteria are vectorised, thus allowing for batch mode search.

Value

A traits object.

Examples

```
## Not run:
# Get a trait by its EFO identifier
get_traits(efo_id = 'EFO_0004631')

# Get a trait by matching a term in EFO identifier (`efo_id`), label,
# description synonyms, categories, or external mapped terms
get_traits(trait_term = 'stroke', exact_term = FALSE)

# Get a trait matching its name (`trait`) exactly (default)
get_traits(trait_term = 'stroke', exact_term = TRUE)

# Get traits, excluding its children traits (default)
get_traits(trait_term = 'breast cancer')

# Get traits, including its children traits (check column `is_child` for
# child traits)
get_traits(trait_term = 'breast cancer', include_children = TRUE)

## End(Not run)
```

`get_trait_categories` *Get PGS Catalog Trait Categories*

Description

Retrieves all trait categories via the PGS Catalog REST API.

Usage

```
get_trait_categories(verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar indicating download progress from the REST API server.

Value

A trait_categories object.

Examples

```
get_trait_categories(progress_bar = FALSE)
```

`n` *Number of PGS Catalog entities*

Description

This function returns the number of entities in a PGS Catalog object. To avoid ambiguity with `dplyr::n()` use `quincunx::n()`.

Usage

```
n(x, unique = FALSE)

## S4 method for signature 'scores'
n(x, unique = FALSE)

## S4 method for signature 'publications'
n(x, unique = FALSE)

## S4 method for signature 'traits'
n(x, unique = FALSE)

## S4 method for signature 'performance_metrics'
n(x, unique = FALSE)

## S4 method for signature 'sample_sets'
n(x, unique = FALSE)

## S4 method for signature 'cohorts'
n(x, unique = FALSE)
```

```
## S4 method for signature 'trait_categories'
n(x, unique = FALSE)

## S4 method for signature 'releases'
n(x, unique = FALSE)
```

Arguments

`x`	A scores, publications, traits, performance_metrics, sample_sets, cohorts, trait_categories or releases object.
`unique`	Whether to count only unique entries (`TRUE`) or not (`FALSE`).

Value

An integer scalar.

Examples

```
# Return the number of polygenic scores in a scores object:
my_scores <- get_scores(pgs_id = c('PGS000007', 'PGS000007', 'PGS000042'))
n(my_scores)

# If you want to count unique scores only, then use the `unique` parameter:
n(my_scores, unique = TRUE)

# Total number of curated publications in the PGS Catalog:
all_pub <- get_publications(interactive = FALSE, progress_bar = FALSE)
n(all_pub)

# Total number of curated traits in the PGS Catalog:
all_traits <- get_traits(interactive = FALSE, progress_bar = FALSE)
n(all_traits)
```

`open_in_dbsnp` *Browse dbSNP from SNP identifiers.*

Description

This function launches the web browser at dbSNP and opens a tab for each SNP identifier.

Usage

```
open_in_dbsnp(variant_id)
```

Arguments

`variant_id`	A variant identifier, a character vector.

Value

Returns `TRUE` if successful. Note however that this function is run for its side effect.

Examples

```
open_in_dbsnp('rs56261590')
```

`open_in_pgs_catalog` *Browse PGS Catalog entities from the PGS Catalog Web Graphical User Interface*

Description

This function launches the web browser and opens a tab for each identifier on the PGS Catalog web graphical user interface: `https://www.pgscatalog.org/`.

Usage

```
open_in_pgs_catalog(
  identifier = NULL,
  pgs_catalog_entity = c("pgs", "pgp", "pss", "efo")
)
```

Arguments

`identifier`	A vector of identifiers. The identifiers can be: PGS, PGP, PSS or EFO identifiers.
`pgs_catalog_entity`	Either `'pgs'` (default), `'pgp'`, `'pss'`, `'efo'`. This argument indicates the type of the identifiers passed in `identifier`.

Value

Returns `TRUE` if successful, or `FALSE` otherwise. But note that this function is run for its side effect.

Examples

```
# Open in PGS scores Catalog Web Graphical User Interface
open_in_pgs_catalog(c('PGS000001', 'PGS000002'))

# Open PGS Catalog Publications
open_in_pgs_catalog(c('PGP000001', 'PGP000002'),
  pgs_catalog_entity = 'pgp')

# Open Sample Sets (PSS)
open_in_pgs_catalog(c('PSS000001', 'PSS000002'),
  pgs_catalog_entity = 'pss')
```

```
# Open EFO traits (EFO)
open_in_pgs_catalog(c('EFO_0001645', 'MONDO_0007254'),
  pgs_catalog_entity = 'efo')
```

`open_in_pubmed` *Browse PubMed from PubMed identifiers.*

Description

This function launches the web browser and opens a tab for each PubMed citation.

Usage

```
open_in_pubmed(pubmed_id)
```

Arguments

`pubmed_id`	A PubMed identifier, either a character or an integer vector.

Value

Returns `TRUE` if successful. Note however that this function is run for its side effect.

Examples

```
open_in_pubmed(c('26301688', '30595370'))
```

`performance_metrics-class` *An S4 class to represent a set of PGS Catalog Performance Metrics*

Description

The performance_metrics object consists of nine tables (slots) that combined form a relational database of a subset of performance metrics. Each performance metric is an observation (row) in the `scores` table (first table).

Slots

`performance_metrics` A table of PGS Performance Metrics (PPM). Each PPM (row) is uniquely identified by the `ppm_id` column. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **pgs_id** Polygenic Score (PGS) identifier.
- **reported_trait** The author-reported trait that the PGS has been developed to predict. Example: `"Breast Cancer"`.
- **covariates** Comma-separated list of covariates used in the prediction model to evaluate the PGS.
- **comments** Any other information relevant to the understanding of the performance metrics.

`publications` A table of publications. Each publication (row) is uniquely identified by the column `pgp_id`. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **pgp_id** PGS Publication identifier. Example: `"PGP000001"`.
- **pubmed_id** PubMed identifier. Example: `"25855707"`.
- **publication_date** Publication date. Example: `"2020-09-28"`. Note that the class of `publication_date` is `Date`.
- **publication** Abbreviated name of the journal. Example: `"Am J Hum Genet"`.
- **title** Publication title.
- **author_fullname** First author of the publication. Example: `'Mavaddat N'`.
- **doi** Digital Object Identifier (DOI). This variable is also curated to allow unpublished work (e.g. preprints) to be added to the catalog. Example: `"10.1093/jnci/djv036"`.

`sample_sets` A table of sample sets. Each sample set (row) is uniquely identified by the column `pss_id`. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **pss_id** A PGS Sample Set identifier. Example: `"PSS000042"`.

`samples` A table of samples. Each sample (row) is uniquely identified by the combination of values from the columns: `ppm_id`, `pss_id`, and `sample_id`. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **pss_id** A PGS Sample Set identifier. Example: `"PSS000042"`.
- **sample_id** Sample identifier. This is a surrogate key to identify each sample.
- **stage** Sample stage: should be always Evaluation (`"eval"`).
- **sample_size** Number of individuals included in the sample.
- **sample_cases** Number of cases.
- **sample_controls** Number of controls.
- **sample_percent_male** Percentage of male participants.
- **phenotype_description** Detailed phenotype description.
- **ancestry_category** Author reported ancestry is mapped to the best matching ancestry category from the NHGRI-EBI GWAS Catalog framework (see `ancestry_categories`) for possible values.
- **ancestry** A more detailed description of sample ancestry that usually matches the most specific description described by the authors (e.g. French, Chinese).

- **country** Author reported countries of recruitment (if available).
- **ancestry_additional_description** Any additional description not captured in the other columns (e.g. founder or genetically isolated populations, or further description of admixed samples).
- **study_id** Associated GWAS Catalog study accession identifier, e.g., `"GCST002735"`.
- **pubmed_id** PubMed identifier.
- **cohorts_additional_description** Any additional description about the samples (e.g. sub-cohort information).

`demographics` A table of sample demographics' variables. Each demographics' variable (row) is uniquely identified by the combination of values from the columns: `ppm_id`, `pss_id`, `sample_id`, and `variable`. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **pss_id** A PGS Sample Set identifier. Example: `"PSS000042"`.
- **sample_id** Sample identifier. This is a surrogate identifier to identify each sample.
- **variable** Demographics variable. Following columns report about the indicated variable.
- **estimate_type** Type of statistical estimate for variable.
- **estimate** The variable's statistical value.
- **unit** Unit of the variable.
- **variability_type** Measure of statistical dispersion for variable, e.g. standard error (se) or standard deviation (sd).
- **variability** The value of the measure of dispersion.
- **interval_type** Type of statistical interval for variable: range, iqr (interquartile), ci (confidence interval).
- **interval_lower** Interval lower bound.
- **interval_upper** Interval upper bound.

`cohorts` A table of cohorts. Each cohort (row) is uniquely identified by the combination of values from the columns: `ppm_id`, `sample_id` and `cohort_symbol`. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **sample_id** Sample identifier. This is a surrogate key to identify each sample.
- **cohort_symbol** Cohort symbol.
- **cohort_name** Cohort full name.

`pgs_effect_sizes` A table of effect sizes per standard deviation change in PGS. Examples include regression coefficients (betas) for continuous traits, odds ratios (OR) and/or hazard ratios (HR) for dichotomous traits depending on the availability of time-to-event data. Each effect size is uniquely identified by the combination of values from the columns: `ppm_id` and `effect_size_id`. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **effect_size_id** Effect size identifier. This is a surrogate identifier to identify each effect size.
- **estimate_type_long** Long notation of the effect size (e.g. Odds Ratio).
- **estimate_type** Short notation of the effect size (e.g. OR).
- **estimate** The estimate's value.
- **unit** Unit of the estimate.

- **variability_type** Measure of statistical dispersion for variable, e.g. standard error (se) or standard deviation (sd).
- **variability** The value of the measure of dispersion.
- **interval_type** Type of statistical interval for variable: range, iqr (interquartile), ci (confidence interval).
- **interval_lower** Interval lower bound.
- **interval_upper** Interval upper bound.

`pgs_classification_metrics` A table of classification metrics. Examples include the Area under the Receiver Operating Characteristic (AUROC) or Harrell's C-index (Concordance statistic). Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **classification_metrics_id** Classification metric identifier. This is a surrogate identifier to identify each classification metric.
- **estimate_type_long** Long notation of the classification metric (e.g. Concordance Statistic).
- **estimate_type** Short notation classification metric (e.g. C-index).
- **estimate** The estimate's value.
- **unit** Unit of the estimate.
- **variability_type** Measure of statistical dispersion for variable, e.g. standard error (se) or standard deviation (sd).
- **variability** The value of the measure of dispersion.
- **interval_type** Type of statistical interval for variable: range, iqr (interquartile), ci (confidence interval).
- **interval_lower** Interval lower bound.
- **interval_upper** Interval upper bound.

`pgs_other_metrics` A table of other metrics that are neither effect sizes nor classification metrics. Examples include: R² (proportion of the variance explained), or reclassification metrics. Columns:

- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.
- **other_metrics_id** Other metric identifier. This is a surrogate identifier to identify each metric.
- **estimate_type_long** Long notation of the metric. Example: "Proportion of the variance explained".
- **estimate_type** Short notation metric. Example: "R²".
- **estimate** The estimate's value.
- **unit** Unit of the estimate.
- **variability_type** Measure of statistical dispersion for variable, e.g. standard error (se) or standard deviation (sd).
- **variability** The value of the measure of dispersion.
- **interval_type** Type of statistical interval for variable: range, iqr (interquartile), ci (confidence interval).
- **interval_lower** Interval lower bound.
- **interval_upper** Interval upper bound.

`pgp_to_pgs` *Map PGP identifiers to PGS identifiers*

Description

Map PGP identifiers to PGS identifiers.

Usage

```
pgp_to_pgs(
  pgp_id = NULL,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgp_id`	A character vector of PGS Catalog Publication identifiers, e.g., "PGP000001". If `NULL` then returns results for all PGP identifiers in the Catalog.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pgp_id` and `pgs_id`.

Examples

```
## Not run:
pgp_to_pgs('PGP000001')
pgp_to_pgs(c('PGP000017', 'PGP000042'))

## End(Not run)
```

`pgp_to_ppm` *Map PGP identifiers to PPM identifiers*

Description

Map PGP identifiers to PPM identifiers.

Usage

```
pgp_to_ppm(
  pgp_id = NULL,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgp_id`	A character vector of PGS Catalog Publication identifiers, e.g., "PGP000001". If `NULL` then returns results for all PGP identifiers in the Catalog.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pgp_id` and `ppm_id`.

Examples

```
## Not run:
pgp_to_ppm('PGP000001')
pgp_to_ppm(c('PGP000017', 'PGP000042'))

## End(Not run)
```

`pgp_to_pss` *Map PGP identifiers to PSS identifiers*

Description

Map PGP identifiers to PSS identifiers.

Usage

```
pgp_to_pss(
  pgp_id = NULL,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgp_id`	A character vector of PGS Catalog Publication identifiers, e.g., "PGP000001". If `NULL` then returns results for all PGP identifiers in the Catalog.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pgp_id` and `pss_id`.

Examples

```
## Not run:
pgp_to_pss('PGP000001')
pgp_to_pss(c('PGP000017', 'PGP000042'))

## End(Not run)
```

`pgs_to_pgp` *Map PGS identifiers to PGP identifiers*

Description

Map PGS identifiers to PGP identifiers.

Usage

```
pgs_to_pgp(
  pgs_id = NULL,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgs_id`	A character vector of PGS identifiers, e.g., "PGS000001". If `NULL` then returns results for all PGS identifiers in the Catalog.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pgs_id` and `pgp_id`.

Examples

```
## Not run:
pgs_to_pgp('PGS000001')
pgs_to_pgp(c('PGS000017', 'PGS000042'))

## End(Not run)
```

`pgs_to_ppm` *Map PGS identifiers to PPM identifiers*

Description

Map PGS identifiers to PPM identifiers.

Usage

```
pgs_to_ppm(pgs_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`pgs_id`	A character vector of PGS identifiers, e.g., "PGS000001".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pgs_id` and `ppm_id`.

Examples

```
## Not run:
pgs_to_ppm('PGS000001')
pgs_to_ppm(c('PGS000017', 'PGS000042'))

## End(Not run)
```

`pgs_to_pss` *Map PGS identifiers to PSS identifiers*

Description

Map PGS identifiers to PSS identifiers.

Usage

```
pgs_to_pss(
  pgs_id = NULL,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgs_id`	A character vector of PGS identifiers, e.g., "PGS000001". If `NULL` then returns results for all PGS identifiers in the Catalog.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pgs_id` and `pss_id`.

Examples

```
## Not run:
pgs_to_pss('PGS000001')
pgs_to_pss(c('PGS000017', 'PGS000042'))

## End(Not run)
```

`pgs_to_study` *Map PGS identifiers to GWAS study identifiers*

Description

Map PGS identifiers to GWAS study identifiers. Retrieves GWAS study identifiers associated with samples used in the discovery stage of queried PGS identifiers.

Usage

```
pgs_to_study(
  pgs_id = NULL,
  verbose = FALSE,
  warnings = TRUE,
  progress_bar = TRUE
)
```

Arguments

`pgs_id`	A character vector of PGS Catalog score accession identifiers., e.g., "PGS000001". If `NULL` then returns results for all PGS identifiers in the Catalog.
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pgs_id` and `study_id`.

Examples

```
## Not run:
pgs_to_study('PGS000001')
# Unmappable pgs ids will be missing, e.g., PGS000023
pgs_to_study(c('PGS000013', 'PGS000023'))

## End(Not run)
```

`ppm_to_pgp` *Map PPM identifiers to PGP identifiers*

Description

Map PPM identifiers to PGP identifiers.

Usage

```
ppm_to_pgp(ppm_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`ppm_id`	A character vector of PPM identifiers, e.g., "PPM000001".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `ppm_id` and `pgp_id`.

Examples

```
## Not run:
ppm_to_pgp('PPM000001')
ppm_to_pgp(c('PPM000017', 'PPM000042'))

## End(Not run)
```

`ppm_to_pgs` *Map PPM identifiers to PGS identifiers*

Description

Map PPM identifiers to PGS identifiers.

Usage

```
ppm_to_pgs(ppm_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`ppm_id`	A character vector of PPM identifiers, e.g., "PPPM000001".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `ppm_id` and `pgs_id`.

Examples

```
## Not run:
ppm_to_pgs('PPM000001')
ppm_to_pgs(c('PPM000017', 'PPM000042'))

## End(Not run)
```

`ppm_to_pss` *Map PPM identifiers to PSS identifiers*

Description

Map PPM identifiers to PSS identifiers.

Usage

```
ppm_to_pss(ppm_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`ppm_id`	A character vector of PPM identifiers, e.g., "PPM000001".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `ppm_id` and `pss_id`.

Examples

```
## Not run:
ppm_to_pss('PPM000001')
ppm_to_pss(c('PPM000017', 'PPM000042'))

## End(Not run)
```

`pss_to_pgp` *Map PSS identifiers to PGP identifiers*

Description

Map PSS identifiers to PGP identifiers. This is a slow function because it starts by downloading first all Performance Metrics, as this is the linkage between PSS and PGP.

Usage

```
pss_to_pgp(pss_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`pss_id`	A character vector of PSS identifiers, e.g., "PSS000001".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pss_id` and `pgp_id`.

Examples

```
## Not run:
pss_to_pgp('PSS000001')
pss_to_pgp(c('PSS000017', 'PSS000042'))

## End(Not run)
```

`pss_to_pgs` *Map PSS identifiers to PGS identifiers*

Description

Map PSS identifiers to PGS identifiers. This is a slow function because it starts by downloading first all Performance Metrics, as this is the linkage between PSS and PGS.

Usage

```
pss_to_pgs(pss_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`pss_id`	A character vector of PSS identifiers, e.g., "PSS000001".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pss_id` and `pgs_id`.

Examples

```
## Not run:
pss_to_pgs('PSS000001')
pss_to_pgs(c('PSS000017', 'PSS000042'))

## End(Not run)
```

`pss_to_ppm` *Map PSS identifiers to PPM identifiers*

Description

Map PSS identifiers to PPM identifiers. This is a slow function because it starts by downloading first all Performance Metrics.

Usage

```
pss_to_ppm(pss_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`pss_id`	A character vector of PSS identifiers, e.g., "PSS000001".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `pss_id` and `ppm_id`.

Examples

```
## Not run:
pss_to_ppm('PSS000001')
pss_to_ppm(c('PSS000017', 'PSS000042'))

## End(Not run)
```

`publications-class` *An S4 class to represent a set of PGS Catalog Publications*

Description

The publications object consists of two tables (slots), each a table that combined form a relational database of a subset of PGS Catalog Publications. Each publication is an observation (row) in the `publications` table (first table).

Slots

`publications` A table of publications. Each publication (row) is uniquely identified by the `pgp_id` column. Columns:

- **pgp_id** PGS Publication identifier. Example: `"PGP000001"`.
- **pubmed_id** PubMed identifier. Example: `"25855707"`.
- **publication_date** Publication date. Example: `"2020-09-28"`. Note that the class of `publication_date` is `Date`.
- **publication** Abbreviated name of the journal. Example: `"Am J Hum Genet"`.
- **title** Publication title.
- **author_fullname** First author of the publication. Example: `'Mavaddat N'`.
- **doi** Digital Object Identifier (DOI). This variable is also curated to allow unpublished work (e.g. preprints) to be added to the catalog. Example: `"10.1093/jnci/djv036"`.
- **authors** Concatenated list of all the publication authors.

`pgs_ids` A table of publication and associated PGS identifiers. Columns:

- **pgp_id** PGS Publication identifier. Example: `"PGP000001"`.
- **pgs_id** Polygenic Score (PGS) identifier.
- **stage** PGS stage: either "gwas/dev" or "eval".

`read_scoring_file` *Read a polygenic scoring file*

Description

This function imports a PGS scoring file. For more information about the scoring file schema check `vignette("pgs-scoring-file", package = "quincunx")`.

Usage

```
read_scoring_file(
  source,
  harmonized = FALSE,
  assembly = c("GRCh38", "GRCh37"),
  protocol = "http",
  metadata_only = FALSE
)
```

Arguments

`source`	PGS scoring file. This can be specified in three forms: (i) a PGS identifier, e.g. `"PGS000001"`, (ii) a path to a local file, e.g. `"~/PGS000001.txt"` or `"~/PGS000001.txt.gz"` or (iii) a direct URL to the PGS Catalog FTP server, e.g. `"http://ftp.ebi.ac.uk/pub/databases/spot/pgs/scores/PGS000001/ScoringFiles/PGS00`
`harmonized`	Whether to read an alternative, harmonized version of the PGS scoring file. This version contains harmonized variant information. This information is provided in extra columns whose names are prefixed with `"hm_"`.

`assembly`	If `harmonized` is `TRUE`, `assembly` indicates which the genome assembly to choose for the harmonized variant data. `assembly` must be either `"GRCh38"` (default) or `"GRCh37"`.
`protocol`	Network protocol for communication with the PGS Catalog FTP server: either `"http"` or `"ftp"`.
`metadata_only`	Whether to read only the comment block (header) from the scoring file.

Value

The returned value is a named list. The names are copied from the arguments passed in `source`. Each element of the list contains another list of two elements: `"metadata"` and `"data"`. The "metadata" element contains data parsed from the header of the PGS scoring file. The "data" element contains a data frame with as many rows as variants that constitute the PGS score. The columns can vary. There are mandatory and optional columns. The mandatory columns are those that identify the variant, effect allele (`effect_allele`), and its respective weight (`effect_weight`) in the score. The columns that identify the variant can either be the `rsID` or the combination of `chr_name` and `chr_position`. The "data" element will be `NULL` is argument `metadata_only` is `TRUE`. For more information about the scoring file schema check `vignette("pgs-scoring-file", package = "quincunx")`.

Examples

```
## Not run:
# Read a PGS scoring file by PGS ID
# (internally, it translates the PGS ID
#  to the corresponding FTP URL)
try(read_scoring_file("PGS000655"))

# Equivalent to `read_scoring_file("PGS000655")`
url <- paste0(
  "http://ftp.ebi.ac.uk/",
  "pub/databases/spot/pgs/scores/",
  "PGS000655/ScoringFiles/",
  "PGS000655.txt.gz"
)
read_scoring_file(url)


# Reading from a local file
try(read_scoring_file("~/PGS000655.txt.gz"))

## End(Not run)
```

`releases-class` *An S4 class to represent a set of PGS Catalog Releases*

Description

The releases object consists of four tables (slots) that combined form a relational database of a subset of PGS Catalog releases. Each release is an observation (row) in the `releases` table (first table).

Slots

`releases` A table of PGS Catalog releases. Each release (row) is uniquely identified by the release date (`date`). Columns:

- **date** Release date.
- **n_pgs** Number of newly released Polygenic Scores.
- **n_ppm** Number of newly released PGS Performance Metrics.
- **n_pgp** Number of newly released PGS Publications.

`pgs_ids` A table of released Polygenic Scores (PGS) identifiers. Columns:

- **date** Release date.
- **pgs_id** Polygenic Score (PGS) identifier. Example: `"PGS000001"`.

`ppm_ids` A table of the released PGS Performance Metrics identifiers. Columns:

- **date** Release date.
- **ppm_id** A PGS Performance Metrics identifier. Example: `"PPM000001"`.

`pgp_ids` A table of the released PGS Publication identifiers. Columns:

- **date** Release date.
- **pgp_id** PGS Publication identifier. Example: `"PGP000001"`.

`sample_sets-class` *An S4 class to represent a set of PGS Catalog Sample Sets*

Description

The sample_sets object consists of four tables (slots) that combined form a relational database of a subset of PGS Catalog sample sets. Each sample set is an observation (row) in the `sample_sets` table (first table).

Slots

`sample_sets` A table of sample sets. Each sample set (row) is uniquely identified by the column `pss_id`. Columns:

- **pss_id** A PGS Sample Set identifier. Example: `"PSS000042"`.

`samples` A table of samples. Each sample (row) is uniquely identified by the combination of values from the columns: `pss_id` and `sample_id`. Columns:

- **pss_id** A PGS Sample Set identifier. Example: `"PSS000042"`.
- **sample_id** Sample identifier. This is a surrogate key to identify each sample.
- **stage** Sample stage: should be always Evaluation (`"eval"`).

- **sample_size** Number of individuals included in the sample.
- **sample_cases** Number of cases.
- **sample_controls** Number of controls.
- **sample_percent_male** Percentage of male participants.
- **phenotype_description** Detailed phenotype description.
- **ancestry_category** Author reported ancestry is mapped to the best matching ancestry category from the NHGRI-EBI GWAS Catalog framework (see `ancestry_categories`) for possible values.
- **ancestry** A more detailed description of sample ancestry that usually matches the most specific description described by the authors (e.g. French, Chinese).
- **country** Author reported countries of recruitment (if available).
- **ancestry_additional_description** Any additional description not captured in the other columns (e.g. founder or genetically isolated populations, or further description of admixed samples).
- **study_id** Associated GWAS Catalog study accession identifier, e.g., `"GCST002735"`.
- **pubmed_id** PubMed identifier.
- **cohorts_additional_description** Any additional description about the samples (e.g. subcohort information).

`demographics` A table of sample demographics' variables. Each demographics' variable (row) is uniquely identified by the combination of values from the columns: `pss_id`, `sample_id`, and `variable`. Columns:

- **pss_id** A PGS Sample Set identifier. Example: `"PSS000042"`.
- **sample_id** Sample identifier. This is a surrogate identifier to identify each sample.
- **variable** Demographics variable. Following columns report about the indicated variable.
- **estimate_type** Type of statistical estimate for variable.
- **estimate** The variable's statistical value.
- **unit** Unit of the variable.
- **variability_type** Measure of statistical dispersion for variable, e.g. standard error (se) or standard deviation (sd).
- **variability** The value of the measure of dispersion.
- **interval_type** Type of statistical interval for variable: range, iqr (interquartile), ci (confidence interval).
- **interval_lower** Interval lower bound.
- **interval_upper** Interval upper bound.

`cohorts` A table of cohorts. Each cohort (row) is uniquely identified by the combination of values from the columns: `pss_id`, `sample_id` and `cohort_symbol`. Columns:

- **pss_id** A PGS Sample Set identifier. Example: `"PSS000042"`.
- **sample_id** Sample identifier. This is a surrogate key to identify each sample.
- **cohort_symbol** Cohort symbol.
- **cohort_name** Cohort full name.

`scores-class` *An S4 class to represent a set of PGS Catalog Polygenic Scores*

Description

The scores object consists of six tables (slots) that combined form a relational database of a subset of PGS Catalog polygenic scores. Each score is an observation (row) in the `scores` table (the first table).

Slots

`scores` A table of polygenic scores. Each polygenic score (row) is uniquely identified by the `pgs_id` column. Columns:

- **pgs_id** Polygenic Score (PGS) identifier. Example: `"PGS000001"`.
- **pgs_name** This may be the name that the authors describe the PGS with in the source publication, or a name that a curator of the PGS Catalog has assigned to identify the score during the curation process (before a PGS identifier has been given). Example: `PRS77_BC`.
- **scoring_file** URL to the scoring file on the PGS FTP server. Example: `"http://ftp.ebi.ac.uk/pub/databases/sp`
- **matches_publication** Indicate if the PGS data matches the published polygenic score (`TRUE`). If not (`FALSE`), the authors have provided an alternative polygenic for the Catalog and some other data, such as performance metrics, may differ from the publication.
- **reported_trait** The author-reported trait that the PGS has been developed to predict. Example: `"Breast Cancer"`.
- **trait_additional_description** Any additional description not captured in the other columns. Example: `"Femoral neck BMD (g/cm2)"`.
- **pgs_method_name** The name or description of the method or computational algorithm used to develop the PGS.
- **pgs_method_params** A description of the relevant inputs and parameters relevant to the PGS development method/process.
- **n_variants** Number of variants used to calculate the PGS.
- **n_variants_interactions** Number of higher-order variant interactions included in the PGS.
- **assembly** The version of the genome assembly that the variants present in the PGS are associated with. Example: `GRCh37`.
- **license** The PGS Catalog distributes its data according to EBI's standard Terms of Use. Some PGS have specific terms, licenses, or restrictions (e.g. non-commercial use) that we highlight in this field, if known.

`publications` A table of publications. Each publication (row) is uniquely identified by the `pgp_id` column. Columns:

- **pgs_id** Polygenic Score (PGS) identifier.
- **pgp_id** PGS Publication identifier. Example: `"PGP000001"`.
- **pubmed_id** PubMed identifier. Example: `"25855707"`.
- **publication_date** Publication date. Example: `"2020-09-28"`. Note that the class of `publication_date` is `Date`.

- **publication** Abbreviated name of the journal. Example: `"Am J Hum Genet"`.
- **title** Publication title.
- **author_fullname** First author of the publication. Example: `'Mavaddat N'`.
- **doi** Digital Object Identifier (DOI). This variable is also curated to allow unpublished work (e.g. preprints) to be added to the catalog. Example: `"10.1093/jnci/djv036"`.

`samples` A table of samples. Each sample (row) is uniquely identified by the combination of values from the columns: `pgs_id` and `sample_id`. Columns:

- **pgs_id** Polygenic score identifier. An identifier that starts with `'PGS'` and is followed by six digits, e.g. `'PGS000001'`.
- **sample_id** Sample identifier. This is a surrogate key to identify each sample.
- **stage** Sample stage: either `"discovery"` or `"training"`.
- **sample_size** Number of individuals included in the sample.
- **sample_cases** Number of cases.
- **sample_controls** Number of controls.
- **sample_percent_male** Percentage of male participants.
- **phenotype_description** Detailed phenotype description.
- **ancestry_category** Author reported ancestry is mapped to the best matching ancestry category from the NHGRI-EBI GWAS Catalog framework (see `ancestry_categories`) for possible values.
- **ancestry** A more detailed description of sample ancestry that usually matches the most specific description described by the authors (e.g. French, Chinese).
- **country** Author reported countries of recruitment (if available).
- **ancestry_additional_description** Any additional description not captured in the other columns (e.g. founder or genetically isolated populations, or further description of admixed samples).
- **study_id** Associated GWAS Catalog study accession identifier, e.g., `"GCST002735"`.
- **pubmed_id** PubMed identifier.
- **cohorts_additional_description** Any additional description about the samples (e.g. subcohort information).

`demographics` A table of sample demographics' variables. Each demographics' variable (row) is uniquely identified by the combination of values from the columns: `pgs_id`, `sample_id` and `variable`. Columns:

- **pgs_id** Polygenic Score (PGS) identifier.
- **sample_id** Sample identifier. This is a surrogate identifier to identify each sample.
- **variable** Demographics variable. Following columns report about the indicated variable.
- **estimate_type** Type of statistical estimate for variable.
- **estimate** The variable's statistical value.
- **unit** Unit of the variable.
- **variability_type** Measure of statistical dispersion for variable, e.g. standard error (se) or standard deviation (sd).
- **variability** The value of the measure of dispersion.
- **interval_type** Type of statistical interval for variable: range, iqr (interquartile), ci (confidence interval).

- **interval_lower** Interval lower bound.
- **interval_upper** Interval upper bound.

`cohorts` A table of cohorts. Each cohort (row) is uniquely identified by the combination of values from the columns: `pgs_id`, `sample_id` and `cohort_symbol`. Columns:

- **pgs_id** Polygenic Score (PGS) identifier.
- **sample_id** Sample identifier. This is a surrogate key to identify each sample.
- **cohort_symbol** Cohort symbol.
- **cohort_name** Cohort full name.

`traits` A table of EFO traits. Each trait (row) is uniquely identified by the combination of the columns `pgs_id` and `efo_id`. Columns:

- **pgs_id** Polygenic Score (PGS) identifier.
- **efo_id** An EFO identifier.
- **trait** Trait name.
- **description** Detailed description of the trait from EFO.
- **url** External link to the EFO entry.

`stages_tally` A table of sample sizes and number of samples sets at each stage.

- **pgs_id** Polygenic Score (PGS) identifier.
- **stage** Sample stage: either `"gwas"`, `"dev"` or `"eval"`.
- **sample_size** Sample size.
- **n_sample_sets** Number of sample sets (only meaningful for the evaluation stage `"eval"`)

`ancestry_frequencies` This table describes the ancestry composition at each stage.

- **pgs_id** Polygenic Score (PGS) identifier.
- **stage** Sample stage: either `"gwas"`, `"dev"` or `"eval"`.
- **ancestry_class_symbol** Ancestry class symbol.
- **frequency** Ancestry fraction (percentage).

`multi_ancestry_composition` A table of a breakdown of the ancestries included in multi-ancestries.

- **pgs_id** Polygenic Score (PGS) identifier.
- **stage** Sample stage: either `"gwas"`, `"dev"` or `"eval"`.
- **multi_ancestry_class_symbol** Multi-ancestry class symbol.
- **ancestry_class_symbol** Ancestry class symbol.

`setop` *Set operations on PGS Catalog objects*

Description

Performs set union, intersection, and (asymmetric!) difference on two objects of either class scores, publications, traits, performance_metrics, sample_sets, cohorts or trait_categories. Note that `union()` removes duplicated entities, whereas `bind()` does not.

Usage

```
union(x, y, ...)

intersect(x, y, ...)

setdiff(x, y, ...)

setequal(x, y, ...)
```

Arguments

`x, y`	Objects of either class scores, publications, traits, performance_metrics, sample_sets, cohorts or trait_categories.
`...`	other arguments passed on to methods.

Value

In the case of `union()`, `intersect()`, or `setdiff()`: an object of the same class as `x` and `y`. In the case of `setequal()`, a logical scalar.

Examples

```
# Get some `scores` objects:
my_scores_1 <- get_scores(c('PGS000012', 'PGS000013'))
my_scores_2 <- get_scores(c('PGS000013', 'PGS000014'))

#
# union()
#
# NB: with `union()`, PGS000013 is not repeated.
union(my_scores_1, my_scores_2)@scores

#
# intersect()
#
intersect(my_scores_1, my_scores_2)@scores

#
# setdiff()
#
setdiff(my_scores_1, my_scores_2)@scores

#
# setequal()
#
setequal(my_scores_1, my_scores_2)
setequal(my_scores_1, my_scores_1)
setequal(my_scores_2, my_scores_2)
```

`stages` *Study stages*

Description

A dataset containing the various study stages assigned to samples in the PGS Catalog.

Usage

```
stages
```

Format

A data frame with 5 stages (rows) and 4 columns:

stage Study stage.

symbol One-letter symbol for the stage, or a comma separated combination thereof.

name Stage name.

definition Stage description.

Source

https://www.pgscatalog.org/docs/ancestry

Examples

```
stages
```

`study_to_pgs` *Map GWAS studies identifiers to PGS identifiers*

Description

Map GWAS studies identifiers to PGS identifiers.

Usage

```
study_to_pgs(study_id, verbose = FALSE, warnings = TRUE, progress_bar = TRUE)
```

Arguments

`study_id`	A character vector of GWAS Catalog study accession identifiers, e.g., "GCST001937".
`verbose`	A `logical` indicating whether the function should be verbose about the different queries or not.
`warnings`	A `logical` indicating whether to print warnings, if any.
`progress_bar`	Whether to show a progress bar as the queries are performed.

Value

A data frame of two columns: `study_id` and `pgs_id`.

Examples

```
## Not run:
study_to_pgs('GCST001937')
study_to_pgs(c('GCST000998', 'GCST000338'))

## End(Not run)
```

`traits-class` *An S4 class to represent a set of PGS Catalog Traits*

Description

The traits object consists of six slots, each a table (`tibble`), that combined form a relational database of a subset of PGS Catalog traits. Each trait is an observation (row) in the `traits` table — main table. All tables have the column `efo_id` as primary key.

Slots

`traits` A table of traits. Columns:

- **efo_id** An EFO identifier.
- **parent_efo_id** An EFO identifier of the parent trait.
- **is_child** Is this trait obtained because it is a child of other trait?
- **trait** Trait name.
- **description** Detailed description of the trait from EFO.
- **url** External link to the EFO entry.

`pgs_ids` A table of associated polygenic score identifiers. Columns:

- **efo_id** An EFO identifier.
- **parent_efo_id** An EFO identifier of the parent trait.
- **is_child** Is this trait obtained because it is a child of other trait?
- **pgs_id** Polygenic Score (PGS) identifier.

`child_pgs_ids` A table of polygenic score identifiers associated with the child traits. Columns:

- **efo_id** An EFO identifier.
- **parent_efo_id** An EFO identifier of the parent trait.
- **is_child** Is this trait obtained because it is a child of other trait?
- **child_pgs_id** Polygenic Score (PGS) identifiers associated with child traits.

`trait_categories` A table of associated trait categories. Columns:

- **efo_id** An EFO identifier.
- **parent_efo_id** An EFO identifier of the parent trait.
- **is_child** Is this trait obtained because it is a child of other trait?

- **trait_category** Trait category name.

`trait_synonyms` A table of associated trait synonyms. Columns:

- **efo_id** An EFO identifier.
- **parent_efo_id** An EFO identifier of the parent trait.
- **is_child** Is this trait obtained because it is a child of other trait?
- **trait_synonyms** Trait synonyms.

`trait_mapped_terms` A table of associated external references, identifiers or other terms. Columns:

- **efo_id** An EFO identifier.
- **parent_efo_id** An EFO identifier of the parent trait.
- **is_child** Is this trait obtained because it is a child of other trait?
- **trait_mapped_terms** Trait mapped terms.

`trait_categories-class`

An S4 class to represent a set of PGS Catalog Trait Categories

---

Description

The trait_categories object consists of two tables (slots) that combined form a relational database of a subset of PGS Catalog trait categories. Each score is an observation (row) in the `trait_categories` table (first table).

Slots

`trait_categories` A table of trait categories. Columns:

- **trait_category** Trait category name.

`traits` A table of associated traits. Columns:

- **trait_category** Trait category name.
- **efo_id** An EFO identifier.
- **trait** Trait name.
- **description** Detailed description of the trait from EFO.
- **url** External link to the EFO entry.

`write_xlsx` *Export a PGS Catalog object to xlsx*

Description

This function exports a PGS Catalog object to Microsoft Excel xlsx file. Each table (slot) is saved in its own sheet.

Usage

```
write_xlsx(x, file = stop("`file` must be specified"))
```

Arguments

`x`	A scores, publications, traits, performance_metrics, sample_sets, cohorts, trait_categories or releases object.
`file`	A file name to write to.

Value

No return value, called for its side effect.

Index