Package 'globaltrends'

July 8, 2026

Type Package

Title Download and Measure Global Trends Through 'Google' Search Volumes

Version 0.2.1

Description 'Google' offers public access to global search volumes from its search engine through the 'Google Trends' portal. The package downloads these search volumes provided by 'Google Trends' and uses them to measure and analyze the distribution of search scores across countries or within countries. The package allows researchers and analysts to use these search scores to investigate global trends based on patterns within these scores. This offers insights such as degree of internationalization of firms and organizations or dissemination of political, social, or technological trends across the globe or within single countries. An outline of the package's methodological foundations and potential applications is available as a working paper: <doi:10.2139/ssrn.3969013>.

License MIT + file LICENSE

URL https://github.com/ha-pu/globaltrends/

BugReports https://github.com/ha-pu/globaltrends/issues/

Depends R (>= 4.1.0)

Imports data.table (>= 1.14.0), gtrendsR (>= 1.5.1), reticulate (>= 1.38), stats (>= 3.5.0), utils

Suggests hms (>= 1.1.0), knitr (>= 1.29), rmarkdown (>= 2.3), testthat (>= 3.0.0), tibble (>= 3.0.0), withr (>= 2.5.0)

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first computations, export, downloads

VignetteBuilder knitr

Encoding UTF-8

Language en-US

LazyData true

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Harald Puhr [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-3308-9553>),
Jakob Muellner [ccp] (ORCID: <https://orcid.org/0000-0002-3443-0469>)

Maintainer Harald Puhr <harald.puhr@gmail.com>

Repository CRAN

Date/Publication 2026-07-08 09:30:07 UTC

Contents

add_control_keyword . . . 3
add_locations . . . 5
add_synonym . . . 6
aggregate_synonyms . . . 7
batch_keywords . . . 8
batch_time . . . 9
compute_doi . . . 10
compute_score . . . 11
countries . . . 13
countries_wdi . . . 14
data_control . . . 15
data_doi . . . 16
data_object . . . 17
data_score . . . 18
disconnect_db . . . 19
download_control . . . 19
download_object . . . 21
download_region . . . 23
download_related . . . 25
export_control . . . 27
get_api_usage . . . 29
gt.env . . . 30
initialize_db . . . 31
initialize_python . . . 32
remove_data . . . 33
start_db . . . 35
us_states . . . 36

Index **37**

`add_control_keyword` *Add batches of control or object keywords*

Description

The function adds one or more batches of keywords with a time period for downloads to the database. The batches serve as input for all download and computation functions.

Usage

```
add_control_keyword(keyword, start_date = "2010-01", end_date = "2020-12")

add_object_keyword(keyword, start_date = "2010-01", end_date = "2020-12")
```

Arguments

`keyword`	Keywords that should be added as batch. Vector of type `character` or a `list` of `character` vectors. The function also allows the usage of codes for search topics instead of search terms.
`start_date`	Start of the time frame for which batch data should be downloaded. Character scalar in the format `"YYYY-MM"`. Defaults to `"2010-01"`.
`end_date`	End of the time frame for which batch data should be downloaded. Character scalar in the format `"YYYY-MM"`. Defaults to `"2020-12"`.

Details

Since Google Trends allows a maximum of five keywords for each query, batches of control keywords can consist of up to five keywords. Since one control keyword is added to batches of object keywords for mapping, object batch length is limited to four keywords. When a `character` vector contains more than four (five) keywords, the vector is split into four-keyword (five-keyword) batches. A `list` must contain `character` vectors of length four (five) or less. Each batch of keywords is combined with a time period for which data will be downloaded. To change the time period for an existing batch, all downloads and computations must be rerun.

Value

Integer vector of the newly added batch IDs (one element per batch created). Batch data is written to tables *batch_keywords* and *batch_time*. A message is printed for each batch.

Warning

If you use search topics for object keywords, make sure to use search topics for control keywords and vice versa. See Google's FAQ for additional information on search topics.

Note

Leading, trailing, and internal whitespace is automatically trimmed from all keywords via `trimws()`.

See Also

- example_keywords()
- example_time()

Examples

```
## Not run:
add_control_keyword(
  keyword = c("gmail", "maps", "translate", "wikipedia", "youtube"),
  start_date = "2016-01", end_date = "2019-12"
)
add_object_keyword(
  keyword = c("apple", "facebook", "google", "microsoft"),
  start_date = "2016-01", end_date = "2019-12"
)

add_control_keyword(
  keyword = c("gmail", "maps", "news", "translate", "weather", "wikipedia", "youtube"),
  start_date = "2016-01", end_date = "2019-12"
)
add_control_keyword(
  keyword = c("amazon", "apple", "facebook", "google", "microsoft", "netflix", "twitter"),
  start_date = "2016-01", end_date = "2019-12"
)

add_control_keyword(
  keyword = list(
    c("gmail", "maps", "news"),
    c("translate", "weather", "wikipedia", "youtube")
  ),
  start_date = "2016-01", end_date = "2019-12"
)
add_control_keyword(
  keyword = list(
    c("amazon", "apple", "facebook", "google"),
    c("microsoft", "netflix", "twitter")
  ),
  start_date = "2016-01", end_date = "2019-12"
)

# search topics
add_control_keyword(
  keyword = c("%2Fm%2F02q_bk", "%2Fm%2F055t58", "%2Fm%2F025sndk", "%2Fm%2F0d07ph", "%2Fm%2F09jcvs"),
  start_date = "2016-01", end_date = "2019-12"
)
# This adds the following topics: Gmail, Google Maps, Google Translate, Wikipedia, YouTube

## End(Not run)
```

`add_locations` *Add a location set*

Description

Adds location codes to a named *location set* in the `data_locations` database table. A location set is a named group of codes (e.g., `"countries"`, `"DACH"`) that is passed as the `locations` argument to download and computation functions. After insertion the set is immediately accessible as `gt.env$<type>`.

Usage

```
add_locations(locations, type, export = TRUE)
```

Arguments

`locations`	Character vector of location codes to add. Each code must appear in `gtrendsR::countries$country_co` (country level) or `gtrendsR::countries$sub_code` (subnational level). Leading/trailing whitespace and duplicates are removed automatically.
`type`	Character scalar. Name of the location set to which `locations` should be added (e.g., `"DACH"`, `"EU"`). After export the set is available as `gt.env$<type>`.
`export`	Logical scalar. If `TRUE` (default), `gt.env` is refreshed so the updated set is available immediately. Set to `FALSE` when calling `add_locations()` several times in sequence to avoid a redundant database read after each call; run `add_locations(..., export = TRUE)` on the final call, or restart the session, to make all sets available.

Details

The package ships with two default sets — `"countries"` and `"us_states"` — written to the database by `start_db()`. Use `add_locations()` to define additional sets such as `"EU"`, `"DACH"`, or subnational regions for a specific country.

The function is idempotent with respect to `(type, location)` pairs: codes that already exist in the named set are silently skipped, so repeated calls are safe. Leading and trailing whitespace is trimmed from all codes before validation and insertion.

Value

Invisibly returns a tibble of the rows appended to `data_locations` (columns: `location`, `type`). Returns a zero-row tibble when all supplied codes already exist in the set. A message is emitted in either case summarising how many codes were added and how many were skipped.

Known API limitation

The Google Trends API cannot handle the location code `"NA"` (Namibia). If `"NA"` is supplied it is dropped with a warning. If it is the only code supplied, the function errors.

See Also

- download_control() and download_object() — pass a location set here
- compute_score() and compute_doi() — pass a location set here
- countries and us_states — built-in location vectors
- start_db() — populates the default "countries" and "us_states" sets
- gtrendsR::countries — source of all valid location codes

Examples

```
## Not run:
# Create a custom set for the DACH region
add_locations(locations = c("AT", "CH", "DE"), type = "DACH")

# Add subnational codes (US states from the built-in vector)
add_locations(locations = us_states, type = "us_states")

# Add several sets without redundant DB reads; refresh once at the end
add_locations(locations = c("AT", "CH", "DE"), type = "DACH", export = FALSE)
add_locations(locations = c("BE", "LU", "NL"), type = "benelux", export = TRUE)

## End(Not run)
```

---

add_synonym *Add synonyms for object keywords*

Description

Registers one or more synonyms for a single object keyword. When compute_score() aggregates search scores, all synonyms are treated as equivalent to the canonical keyword and their scores are summed.

A common use-case is alternate names: e.g., "FC Bayern" and "Bayern Munich" refer to the same entity and should be aggregated.

Usage

```
add_synonym(keyword, synonym)
```

Arguments

keyword	Character scalar. The canonical object keyword for which the synonyms are registered. Must already exist as an object keyword in the database.
synonym	Character scalar or vector, or a list of character vectors. One or more synonyms to associate with keyword. Each element is inserted as a separate row in *keyword_synonyms*.

Value

Invisibly returns NULL. Synonym rows are written to table *keyword_synonyms* and the in-memory cache gt.env$keyword_synonyms is refreshed. A message is printed for each synonym added.

Note

trimws() is applied to both keyword and synonym to remove leading, trailing, and internal whitespace.

See Also

- compute_score()

Examples

```
## Not run:
# Single synonym
add_synonym(
  keyword = "fc bayern",
  synonym = "bayern munich"
)

# Multiple synonyms in one call
add_synonym(
  keyword = "fc barcelona",
  synonym = c("barcelona", "barca", "fcb")
)

## End(Not run)
```

---

aggregate_synonyms *Aggregate search scores across synonym terms*

Description

Merges synonym keyword scores into their canonical keyword scores in data_score. Run this after compute_score(). Synonym relationships are defined with add_synonym().

Usage

```
aggregate_synonyms(control)
```

Arguments

control	Numeric/integer scalar. The control batch id (batch_c), identifying the reference search used for score normalisation. In most single-control setups this is 1.

Details

For a given `control` batch (`batch_c`), this function:

1. Retrieves all canonical-synonym pairs and their associated object batches (`batch_o`).
2. Pulls the relevant `data_score` rows, remaps synonym rows onto their canonical keyword, and sums scores across duplicates.
3. Deletes the affected `data_score` rows for those object batches.
4. Writes the aggregated rows back to `data_score`.

Value

Invisibly returns a data frame of the rows written to `data_score`. Called primarily for its side effects (data modifications).

See Also

`compute_score()` to populate `data_score` before aggregating, `add_synonym()` to define synonym relationships,

Examples

```
## Not run:
compute_score(object = 1:2, control = 1)
aggregate_synonyms(control = 1)

## End(Not run)
```

---

batch_keywords *Example table: keyword batches (*`batch_keywords`*)*

Description

Example data representing the database table `batch_keywords`. Each row assigns a single `keyword` to a `batch` and a `type` (`"control"` or `"object"`).

The example contains one control batch (5 keywords: gmail, maps, translate, wikipedia, youtube) and four object batches (14 object keywords covering football clubs and technology firms), all covering the period 2010-01 to 2019-12.

In a live database, keyword batches are created via `add_keyword()` and are exported to the package environment `gt.env` by `start_db()` as `gt.env$keywords_control` and `gt.env$keywords_object`. Control batches hold up to five keywords; object batches hold up to four (one slot is reserved for the overlap keyword used in score mapping).

Usage

```
example_keywords
```

Format

A tibble with 3 variables:

type Character. Batch type: `"control"` or `"object"`.

batch Integer. Batch identifier within `type`.

keyword Character. Keyword assigned to the batch.

See Also

`add_keyword()`, `start_db()`

---

batch_time *Example table: batch time windows (`batch_time`)*

Description

Example data representing the database table `batch_time`. Each row assigns a time window (`start_date`, `end_date`) to a `batch` and a `type` (`"control"` or `"object"`). Each `(type, batch)` combination has exactly one row.

In a live database, batch time windows are generated when keywords are added (see `add_keyword()`) and are exported to the package environment `gt.env` by `start_db()` as `gt.env$time_control` and `gt.env$time_object`.

Dates are stored as `"YYYY-MM"` strings to represent monthly windows. To change the time window for an existing batch, all downloads and computations for that batch must be re-run.

Usage

```
example_time
```

Format

A tibble with 4 variables:

type Character. Batch type: `"control"` or `"object"`.

batch Integer. Batch identifier within `type`.

start_date Character. Window start in `"YYYY-MM"`.

end_date Character. Window end in `"YYYY-MM"`.

See Also

`add_keyword()`, `start_db()`

compute_doi *Compute degree of internationalization (DOI)*

Description

Computes degree of internationalization (DOI) for object keywords based on the cross-location distribution of search scores. DOI is computed per (keyword, date) combination for a given control batch (batch_c), object batch (batch_o), and a named location set (e.g., "countries"). Results are appended to the data_doi database table.

Usage

```
compute_doi(object, control = 1, locations = "countries")

## S3 method for class 'numeric'
compute_doi(object, control = 1, locations = "countries")

## S3 method for class 'list'
compute_doi(object, control = 1, locations = "countries")
```

Arguments

object	Numeric scalar, vector, or list of numerics. One or more object batch ids (batch_o) identifying keyword groups for which DOI should be computed. A numeric vector is processed element-by-element (equivalent to passing a list).
control	Numeric scalar. Control batch id (batch_c) identifying the baseline keyword group used for score normalisation. Defaults to 1.
locations	Character scalar. Name of a location set stored in data_locations$type (e.g., "countries", "us_states"). Only locations belonging to this set are included in the DOI computation. Defaults to "countries".

Details

DOI captures how evenly search interest is spread across a set of locations: a perfectly uniform score vector yields the maximum DOI, while one concentrated in a single location yields the minimum.

Three complementary dispersion measures are computed for each (keyword, date) series:

gini 1 - Gini(score). Uses the rank-weighted formula Gini = (2 * sum(score[i] * i) / sum(score) - (n + 1)) / n over the sorted score vector. Ranges from 0 (complete concentration) to 1 (perfect equality).

hhi 1 - HHI(score) where HHI = sum(p^2) and p = score / sum(score). Ranges from 0 (monopoly) to 1 - 1/n (perfect equality across n locations).

entropy H(p) - log(n) where p = score / sum(score), H(p) = -sum(p * log(p)) is Shannon entropy, and n is the number of locations with non-zero scores. Always <= 0; equals 0 when scores are perfectly uniform and becomes more negative as concentration increases. Zero scores are excluded before computing logs.

If all scores for a (keyword, date) series are NA, all three measures are set to NA. If all non-NA scores are zero, gini and hhi return 0 and entropy returns 0.

Score data must already exist in data_score, typically produced by compute_score(). Only locations whose type in data_locations matches the locations argument are included. The global aggregate (location == "world") is excluded unless the location set explicitly contains it.

If DOI for the requested (batch_c, batch_o, locations) combination already exists in data_doi, the function emits a message and returns early without recomputing.

Value

Invisibly returns the data frame appended to data_doi for the processed batch, with columns date, keyword, gini, hhi, entropy, batch_c, batch_o, and locations. Returns an empty data frame when DOI already exists or when no matching score data is found. Called primarily for its side effects (database writes) and emits a progress message per batch.

See Also

compute_score() to produce the score data consumed by this function; data_doi for the database table schema.

Examples

```
## Not run:
compute_doi(object = 1, control = 1, locations = "countries")
compute_doi(object = as.list(1:5), control = 1, locations = "countries")

## End(Not run)
```

---

compute_score *Compute search scores for object keywords*

Description

Computes *search scores* for object keywords by mapping object and control search volumes onto a common scale and then normalizing object volumes by the mapped control total for each (location, date).

Convenience wrapper around compute_score() for computing the *volume of internationalization* (VOI) — a measure of how globally distributed search interest for a keyword is relative to the control baseline. Equivalent to compute_score(object, control, locations = "world"), which uses the worldwide aggregate rather than country-level breakdowns.

Use this function when you only need the global aggregate score, for example when locations = "world" was passed to download_object().

Usage

```
compute_score(object, control = 1, locations = NULL)

## S3 method for class 'numeric'
compute_score(object, control = 1, locations = NULL)

## S3 method for class 'list'
compute_score(object, control = 1, locations = NULL)

compute_voi(object, control = 1)
```

Arguments

`object`	Integer-like scalar, vector, or list. The object batch id(s) (`batch_o`) for which VOI should be computed.
`control`	Integer-like scalar. The control batch id (`batch_c`). Defaults to `1`.
`locations`	Character vector of location codes to compute scores for. The package exports `countries` (ISO 3166-1 alpha-2 codes for all countries) and `us_states` (two-letter US state codes) as convenience vectors. Pass `"world"` to compute the global aggregate only (see also `compute_voi()`). If `NULL`, defaults to `gt.env$countries` when set via `start_db()`, otherwise falls back to `globaltrends::countries`.

Details

Conceptually, the score for an object keyword is computed as:

$$score_{o,loc,t} = \frac{hits_{o,loc,t}}{\sum_{k \in C} \tilde{hits}_{k,loc,t}}$$

where C is the set of control keywords and $\tilde{hits}$ are control hits mapped to the object scale using an overlap-based benchmark, following the mapping logic described in Castelnuovo and Tran (2017, Appendix A).

Idempotency. Already-computed (`batch_c, batch_o, location`) combinations are detected and skipped automatically, so repeated calls safely fill in only missing locations.

Operationally, for each object batch (`batch_o`) and control batch (`batch_c`), the function:

1. Identifies the subset of `locations` not yet present in `data_score` for this (`batch_c, batch_o`) pair.
2. Computes a per-(`location, date`) *benchmark* as the mean ratio of object-to-control hits for the keywords that appear in both downloads.
3. Maps control hits to the object scale: `hits_mapped` = `hits` * `benchmark`.
4. Sums mapped control hits across keywords to obtain `hits_c` and computes `score` = `hits_object` / `hits_c` for each object keyword.
5. Inserts the resulting rows into `data_score`.

If synonym keywords were specified via `add_synonym()`, run `aggregate_synonyms()` after score computation to roll synonym scores into their canonical terms.

Value

Called primarily for its side effects (writing to `data_score`); the return value is rarely needed. When `object` is a scalar or vector, returns the number of rows inserted into `data_score` as an integer (`0L` if all requested locations were already computed). When `object` is a list, returns `TRUE` invisibly after processing all elements.

See `compute_score()` for return value semantics.

References

Castelnuovo, E. & Tran, T. D. (2017). Google It Up! A Google Trends-based Uncertainty index for the United States and Australia. *Economics Letters*, *161*, 149–153. doi:10.1016/j.econlet.2017.09.032

See Also

`download_control()` and `download_object()` to populate the raw data tables before computing scores. `aggregate_synonyms()` to roll synonym keyword scores into their canonical terms after score computation. `add_synonym()` to define synonym relationships. `compute_voi()` for the global-aggregate shorthand.

`compute_score()` for country-level scores.

Examples

```
## Not run:
# Compute scores for a single object batch across all countries
compute_score(object = 1, control = 1, locations = countries)

# Process multiple object batches in one call
compute_score(object = as.list(1:5), control = 1, locations = countries)

# Compute the global aggregate (VOI) only
compute_voi(object = 1, control = 1)

## End(Not run)
```

---

countries *Default location set: countries*

Description

Character vector of country location codes used by the package as a default location set for cross-country computations.

The vector contains ISO 3166-1 alpha-2 country codes selected from countries_wdi based on a GDP share threshold (>= 0.1% of world GDP in 2018) using World Bank World Development Indicators (WDI). This threshold retains the economically significant countries while keeping query volume manageable. Pass this vector as the `locations` argument to `compute_score()` or `compute_doi()` for standard cross-country analyses.

Note that "NA" (Namibia's ISO code) is excluded because the Google Trends API cannot handle it; see add_locations() for details.

Usage

```
countries
```

Format

A character vector of ISO 3166-1 alpha-2 country codes.

See Also

countries_wdi, add_locations(), start_db()

Examples

```
length(countries)
head(countries)
```

---

countries_wdi *Country codes and names from WDI*

Description

A data frame of country/location codes and names as provided by the World Bank World Development Indicators (WDI). This object is a bundled snapshot of WDI::WDI_data$country included to remove the runtime dependency on the WDI package. It is useful for mapping ISO-style codes to human-readable country names when inspecting or constructing custom location sets, and for understanding which countries are included in countries.

Usage

```
countries_wdi
```

Format

A data frame whose columns follow the conventions of WDI::WDI_data$country. Key columns include iso2c (ISO 3166-1 alpha-2 code, matching values in countries), country (English country name), and additional World Bank metadata fields.

Source

World Bank World Development Indicators (WDI), https://datatopics.worldbank.org/world-development-indicat
Bundled as a static snapshot; for the latest data see the WDI R package.

See Also

countries, add_locations()

data_control *Example table: control downloads (*data_control*)*

Description

Example data representing the database table data_control. Each row contains Google Trends hits for a control keyword in a given location on a given date, along with the control batch identifier.

In a live database, data are downloaded via download_control() and are accessible via gt.env$dt_* tables after start_db(). Global aggregates use "world" as location.

The example dataset is simulated to resemble real Google Trends output. Simulated values are bounded to the empirical [min, max] range observed in actual downloads for each keyword–location pair.

Usage

```
example_control
```

Format

A tibble with 5 variables:

location Character. Location code (ISO 3166-1 alpha-2 or other codes supported by Google Trends). Global data uses "world".

keyword Character. Control keyword.

date Integer. Date stored as days since 1970-01-01 (Unix epoch). Convert with as.Date(date, origin = "1970-01-01").

hits Integer. Relative search interest in [0, 100]. Google Trends normalizes all values within a single query window so the peak observation equals 100.

batch Integer. Control batch id.

Source

Google Trends (https://trends.google.com/trends/). Simulated to match empirical distributional statistics from real downloads.

See Also

download_control(), start_db()

data_doi *Example table: degree of internationalization (*data_doi*)*

Description

Example data representing the database table data_doi. Each row contains degree-of-internationalization (DOI) metrics for an object keyword on a given date, computed from the distribution of data_score across a specified set of locations.

DOI captures how evenly search interest is spread across locations: a perfectly uniform score distribution yields the maximum value for each metric; concentration in a single location yields the minimum. Three complementary dispersion measures are provided — see compute_doi() for their exact formulae.

DOI is computed via compute_doi() and is queryable through gt.env$dt_doi after start_db(). The batch_c column indicates the control batch used as baseline, and batch_o indicates the object batch.

The example dataset is simulated to resemble outputs derived from real Google Trends data.

Usage

```
example_doi
```

Format

A tibble with 8 variables:

keyword Character. Object keyword.

date Integer. Date stored as days since 1970-01-01. Convert with as.Date(date, origin = "1970-01-01").

gini Double. 1 - Gini(score) across locations. Range [0, 1]: 1 = perfectly equal distribution; 0 = all search interest in one location.

hhi Double. 1 - HHI(score) across locations. Range [0, 1 - 1/n] where n is the number of locations: higher values indicate more equal distributions.

entropy Double. H(p) - log(n) (Shannon entropy deficit). Range (-Inf, 0]: 0 = perfectly uniform distribution; more negative values indicate greater concentration.

batch_c Integer. Control batch id used as baseline.

batch_o Integer. Object batch id.

locations Character. Name of the location set used (e.g., "countries", "us_states").

References

Castelnuovo, E. & Tran, T. D. (2017). Google It Up! A Google Trends-based Uncertainty index for the United States and Australia. *Economics Letters*, *161*, 149–153. doi:10.1016/j.econlet.2017.09.032

Puhr, H. & Müllner, J. (2022). Let me Google that for you: Capturing internationalization using Google Trends. Available at SSRN: doi:10.2139/ssrn.3969013

See Also

compute_doi(), start_db()

---

data_object *Example table: object downloads (*data_object*)*

Description

Example data representing the database table data_object. Each row contains Google Trends hits for an object keyword in a given location on a given date. Each download pairs an object batch (batch_o) with a control batch (batch_c): one control keyword is included in every object query so that object and control hits can be mapped onto a common scale during score computation.

In a live database, data are downloaded via download_object() and are accessible via gt.env$dt_* tables after start_db(). Global aggregates use "world" as location.

The example dataset is simulated to resemble real Google Trends output. Simulated values are bounded to the empirical [min, max] range observed in actual downloads for each keyword–location pair.

Usage

```
example_object
```

Format

A tibble with 6 variables:

location Character. Location code. Global data uses "world".

keyword Character. Object keyword.

date Integer. Date stored as days since 1970-01-01. Convert with as.Date(date, origin = "1970-01-01").

hits Integer. Relative search interest in [0, 100] within the query window. The peak value across all keywords in that query equals 100.

batch_c Integer. Control batch id. Identifies which control batch was co-downloaded for scale mapping in compute_score().

batch_o Integer. Object batch id.

Source

Google Trends (https://trends.google.com/trends/). Simulated to match empirical distributional statistics from real downloads.

See Also

download_object(), start_db()

`data_score` *Example table: computed scores (*`data_score`*)*

Description

Example data representing the database table `data_score`. Each row contains a computed `score` for an object `keyword` in a given `location` on a given `date`, along with the associated control batch (`batch_c`) and object batch (`batch_o`).

Scores are computed by `compute_score()` as:

$$score = \frac{hits_o}{\sum_{k \in C} \tilde{hits}_k}$$

where $hits_o$ are object search volumes and $\tilde{hits}_k$ are control keyword hits mapped to the object scale via an overlap-based benchmark (see Castelnuovo & Tran, 2017). Scores are non-negative; values greater than 1 are possible when object interest exceeds control interest.

In a live database, scores are queryable through `gt.env$dt_scores` after `start_db()`. Global aggregates use `"world"` as `location`.

The example dataset is simulated to resemble outputs derived from real Google Trends data.

Usage

```
example_score
```

Format

A tibble with 6 variables:

location Character. Location code. Global data uses `"world"`.

keyword Character. Object keyword.

date Integer. Date stored as days since 1970-01-01. Convert with `as.Date(date, origin = "1970-01-01")`.

score Double. Normalised search interest (object hits divided by total mapped control hits). Non-negative; 0 when no control data are available.

batch_c Integer. Control batch id used as baseline.

batch_o Integer. Object batch id.

References

Castelnuovo, E. & Tran, T. D. (2017). Google It Up! A Google Trends-based Uncertainty index for the United States and Australia. *Economics Letters*, *161*, 149–153. doi:10.1016/j.econlet.2017.09.032

See Also

`compute_score()`, `start_db()`

disconnect_db *Disconnect from the database and persist changes*

Description

Exports the current in-memory state to the RDS store under `db/` and clears table handles from `gt.env`.

Usage

```
disconnect_db()
```

Details

Call this function after all downloads and computations are complete. It overwrites `db/globaltrends.rds` with the current in-memory state. All `dt_*` handles in `gt.env` are set to `NULL` afterwards.

Data modified during the session will be **lost** if this function is not called before the R session ends.

Value

Invisibly returns `TRUE`. Called for its side effects (writing files under `db/` and clearing handles).

See Also

`initialize_db()` to create the store; `start_db()` to open a new session.

Examples

```
## Not run:
start_db()
# ... downloads and computations ...
disconnect_db()

## End(Not run)
```

download_control *Download data for control keyword batches*

Description

Downloads Google Trends search volumes for one or more *control* batches across a set of locations and appends the results to the database table `data_control`.

Convenience wrapper around `download_control()` that downloads the worldwide aggregate instead of country-level data. Equivalent to calling `download_control(control, locations = "world")`.

Usage

```
download_control(control, locations = NULL)

## S3 method for class 'numeric'
download_control(control, locations = NULL)

## S3 method for class 'list'
download_control(control, locations = NULL)

download_control_global(control)
```

Arguments

control	Numeric scalar, numeric vector, or list of numeric scalars. Control batch id(s) to download.
locations	Character vector of ISO 3166-1 alpha-2 location codes. Defaults to gt.env$countries when set by start_db(); otherwise falls back to globaltrends::countries. Pass "world" (or use download_control_global()) to download the world-wide aggregate instead of country-level data.

Details

Prerequisites. start_db() must be called before download_control(). It connects to the database and populates gt.env$keywords_control and gt.env$time_control from the tables batch_keywords and batch_time (created via add_keyword()). These in-memory objects are used to look up the keywords and time window for each requested batch.

Dispatch. download_control() is an S3 generic that dispatches on the class of control. Passing a numeric scalar routes to the .numeric method, which performs the actual download. Passing a numeric vector of length > 1 coerces control to a list and delegates to the .list method, which iterates over batches sequentially. Passing a list directly also routes to the .list method.

Download backend. Requests are issued through the internal .get_trend() helper, which uses either gtrendsR::gtrends() (default) or the Google Trends Research API when initialize_python() has been called.

Deduplication. Before downloading, the function queries data_control for locations already present for the requested batch. Only locations not yet in the database are downloaded. If all locations are already present, the function returns early with a message and no requests are made.

Missing data. If the API returns no data for a location (e.g. due to insufficient search volume), the result for that location is silently skipped (nothing is written to data_control) and a "No data returned" message is emitted.

Value

Invisibly returns TRUE. The function is called for its side effects: downloaded rows are appended to data_control in the active database, and one progress message is emitted per location indicating whether data was written or no data was returned.

Invisibly returns TRUE. See download_control() for details on side effects and emitted messages.

Category codes

Avoid category codes unless you are confident they apply uniformly to all keywords in the batch. Google Trends applies a category constraint to the entire request, which can unintentionally change the meaning of control and object keywords.

See Also

`start_db()` to connect to the database and populate `gt.env`. `add_keyword()` to register control batches before downloading. `download_control_global()` for a convenience wrapper for worldwide data. `download_object()` to download object keyword data using a control batch for scaling.

Examples

```
## Not run:
# Download one control batch for all countries
download_control(control = 1, locations = countries)

# Download several batches sequentially
download_control(control = as.list(1:5), locations = countries)

# Download worldwide aggregate
download_control_global(control = 1)

## End(Not run)
```

---

download_object *Download data for object keyword batches*

Description

Downloads Google Trends search volumes for one or more *object* batches across a set of locations and appends the results to the database table `data_object`. Each object batch is downloaded together with one control keyword so that object hits can be mapped to the control scale used elsewhere in the package.

Convenience wrapper around `download_object()` that downloads the worldwide aggregate instead of country-level data. Equivalent to calling `download_object(object, control, locations = "world")`.

Usage

```
download_object(object, control = 1, locations = NULL)

## S3 method for class 'numeric'
download_object(object, control = 1, locations = NULL)

## S3 method for class 'list'
```

```
download_object(object, control = 1, locations = NULL)

download_object_global(object, control = 1)
```

Arguments

object	Numeric scalar, numeric vector, or list of numeric scalars. Object batch id(s) to download.
control	Numeric scalar. Control batch id used for mapping. Defaults to 1.
locations	Character vector of ISO 3166-1 alpha-2 location codes. Defaults to gt.env$countries when set by start_db(); otherwise falls back to globaltrends::countries. Pass "world" (or use download_object_global()) to download the world-wide aggregate instead of country-level data.

Details

Prerequisites. start_db() must be called before download_object(). It connects to the database and populates gt.env$keywords_object and gt.env$time_object from the tables batch_keywords and batch_time (created via add_keyword()). These in-memory objects are used to look up the keywords and time window for each requested batch. data_control for the chosen control batch must also be present, as it is used to select an appropriate control keyword per location.

Dispatch. download_object() is an S3 generic that dispatches on the class of object. Passing a numeric scalar routes to the .numeric method, which performs the actual download. Passing a numeric vector of length > 1 coerces object to a list and delegates to the .list method, which iterates over batches sequentially. Passing a list directly also routes to the .list method.

Control keyword selection. For each location the function queries data_control for the chosen control batch, ranks control keywords by their average hits in ascending order, and tries them one by one until one yields non-zero signal in the returned series. Trying lower-signal keywords first reduces saturation risk. If no control keyword produces usable signal, the function stops with an informative error.

Download backend. Requests are issued through the internal .get_trend() helper, which uses either gtrendsR::gtrends() (default) or the Google Trends Research API when initialize_python() has been called.

Deduplication. Before downloading, the function queries data_object for locations already present for the requested (batch_c, batch_o) pair. Only locations not yet in the database are downloaded. If all locations are already present, the function returns early with a message and no requests are made.

Missing control baseline. If data_control contains no rows for a given location, that location is skipped with a message (nothing is written to data_object).

Value

Invisibly returns TRUE. The function is called for its side effects: downloaded rows are appended to data_object in the active database, and one progress message is emitted per location. Locations with no control baseline in data_control are skipped with a message.

Invisibly returns TRUE. See download_object() for details on side effects and emitted messages.

Category codes

Avoid category codes unless you are confident they apply uniformly to all keywords in the batch. Google Trends applies a category constraint to the entire request, which can unintentionally change the meaning of control and object keywords.

See Also

`start_db()` to connect to the database and populate `gt.env`. `add_keyword()` to register object batches before downloading. `download_object_global()` for a convenience wrapper for worldwide data. `download_control()` to download control keyword data used for scaling.

Examples

```
## Not run:
# Download one object batch for all countries
download_object(object = 1, control = 1, locations = countries)

# Download several batches sequentially
download_object(object = as.list(1:5), control = 1, locations = countries)

# Download worldwide aggregate
download_object_global(object = 1, control = 1)

## End(Not run)
```

---

download_region *Download regional interest data for object keywords*

Description

Downloads regional interest data (sub-geo breakdown) for the keywords in one or more object batches (`batch_o`) and writes the results to the database table `data_region`.

Convenience wrapper around `download_region()` that downloads the worldwide aggregate instead of country-level data. Equivalent to calling `download_region(object, locations = "world")`.

Usage

```
download_region(object, locations = NULL)

## S3 method for class 'numeric'
download_region(object, locations = NULL)

## S3 method for class 'list'
download_region(object, locations = NULL)

download_region_global(object)
```

Arguments

object	Numeric scalar, numeric vector, or list of numeric scalars. Object batch id(s) to download.
locations	Character vector of location codes. Defaults to gt.env$countries when set by start_db(); otherwise falls back to globaltrends::countries. Pass "world" (or use download_region_global()) to download the worldwide aggregate instead of country-level data.

Details

Prerequisites. initialize_python() must be called before download_region() to initialise the Research API backend. start_db() must also have been called to connect to the database and populate gt.env$keywords_object and gt.env$time_object.

Dispatch. download_region() is an S3 generic that dispatches on the class of object. Passing a numeric scalar routes to the .numeric method, which performs the actual download. Passing a numeric vector of length > 1 coerces object to a list and delegates to the .list method, which iterates over batches sequentially. Passing a list directly also routes to the .list method.

Download backend. Requests are issued through the internal .get_region() helper using the Google Trends Research API. This backend always requires Python to be set up via initialize_python(); unlike download_control(), no gtrendsR fallback is available.

Deduplication. Before downloading, the function queries data_region for locations already present for the requested object batch. Only locations not yet in the database are downloaded. If all requested locations are already present, the function returns early with a message and no requests are made.

Missing data. If the API returns no data for a location (e.g. due to insufficient search volume), the result for that location is silently skipped (nothing is written to data_region) and a "No region data returned" message is emitted.

Value

Invisibly returns TRUE. The function is called for its side effects: downloaded rows are appended to data_region in the active database, and one progress message is emitted per location indicating whether data was written or no data was returned.

Invisibly returns TRUE. See download_region() for details on side effects and emitted messages.

See Also

initialize_python() to set up the Python backend before downloading. start_db() to connect to the database and populate gt.env. add_keyword() to register object batches before downloading. download_region_global() for a convenience wrapper for worldwide data. download_control() to download control keyword data.

Examples

```
## Not run:
# Download one object batch for all countries
initialize_python(api_key = "XXX", conda_env = "/path/to/env")
```

```
start_db()
download_region(object = 1, locations = countries)

# Download several batches sequentially
download_region(object = as.list(1:3), locations = countries)

# Download worldwide aggregate
download_region_global(object = 1)

## End(Not run)
```

`download_related` *Download related topics and themes for object keyword batches*

Description

Downloads Google Trends related topics or themes for one or more *object* batches across a set of locations and appends the results to the database table `data_related`. Convenience wrappers cover all four combinations of `topic` (topics vs. themes) and `rising` (top vs. rising):

- `download_topics()` / `download_topics_global()` — top related topics
- `download_themes()` / `download_themes_global()` — top related themes
- `download_topics_rising()` / `download_topics_rising_global()` — rising related topics
- `download_themes_rising()` / `download_themes_rising_global()` — rising related themes

Usage

```
download_related(object, locations = NULL, topic = NULL, rising = NULL)

download_topics(object, locations = NULL)

download_themes(object, locations = NULL)

download_topics_rising(object, locations = NULL)

download_themes_rising(object, locations = NULL)

download_topics_global(object)

download_themes_global(object)

download_topics_rising_global(object)

download_themes_rising_global(object)

## S3 method for class 'numeric'
```

```
download_related(object, locations = NULL, topic = NULL, rising = NULL)

## S3 method for class 'list'
download_related(object, locations = NULL, topic = NULL, rising = NULL)
```

Arguments

object	Numeric scalar, numeric vector, or list of numeric scalars. The object batch id(s) to download. A scalar downloads a single batch; a vector or list downloads multiple batches sequentially. Must refer to batches already registered via add_keyword().
locations	Character vector of ISO 3166-1 alpha-2 location codes. Defaults to gt.env$countries when set by start_db(); otherwise falls back to globaltrends::countries. Pass "world" (or use the _global convenience wrappers) to download the worldwide aggregate instead of country-level data.
topic	Logical scalar. TRUE to download related *topics*; FALSE to download related *themes*. The convenience wrappers set this automatically.
rising	Logical scalar. TRUE to download *rising* (breakout) results; FALSE to download *top* results. The convenience wrappers set this automatically.

Details

Prerequisites. initialize_python() must be called before download_related() to set up the Research API backend. start_db() must also have been called; it connects to the database and populates gt.env$keywords_object and gt.env$time_object from the tables batch_keywords and batch_time (created via add_keyword()). These in-memory objects are used to look up the keywords and time window for each requested batch.

Dispatch. download_related() is an S3 generic that dispatches on the class of object. Passing a numeric scalar routes to the .numeric method, which performs the actual download. Passing a numeric vector of length > 1 coerces object to a list and delegates to the .list method, which iterates over batches sequentially. Passing a list directly also routes to the .list method.

Download backend. Requests are issued through the internal .get_related() helper, which queries the Google Trends Research API via the Python backend. One API call is made per keyword per location; results across keywords are row-bound before being written to data_related.

Deduplication. Before downloading, the function queries data_related for locations already present for the requested batch (filtered by topic and rising). Only locations not yet in the database are downloaded. If all locations are already present, the function returns early with a message and no requests are made.

Missing data. If the API returns no data for a location (e.g. due to insufficient search volume), the result for that location is skipped (nothing is written to data_related) and a "No data returned" message is emitted.

Value

Invisibly returns TRUE. The function is called for its side effects: downloaded rows are appended to data_related in the active database, and one progress message is emitted per location indicating whether data was written or no data was returned.

See Also

initialize_python() to set up the Python backend (required before use). start_db() to connect to the database and populate gt.env. add_keyword() to register object batches before downloading. download_object() to download raw search volume data for object keywords.

Examples

```
## Not run:
# Download top related topics for one object batch across all countries
download_topics(object = 1, locations = countries)

# Download rising related themes for several batches sequentially
download_themes_rising(object = as.list(1:5), locations = countries)

# Download top related topics worldwide
download_topics_global(object = 1)

## End(Not run)
```

---

export_control *Export data from database tables*

Description

Seven functions for exporting filtered subsets of the four computed data tables. Each function returns a data frame that can be passed directly to standard R I/O functions such as readr::write_csv() or writexl::write_xlsx().

Function	Source table	Location scope
export_control()	data_control (control hits)	country/region level
export_control_global()	data_control	world aggregate only
export_object()	data_object (object hits)	country/region level
export_object_global()	data_object	world aggregate only
export_score()	data_score (normalized scores)	country/region level
export_voi()	data_score	world aggregate only (VOI)
export_doi()	data_doi (internationalization)	aggregated across locations

Usage

```
export_control(control = NULL, location = NULL)

export_control_global(control = NULL)

export_object(keyword = NULL, object = NULL, control = NULL, location = NULL)

export_object_global(keyword = NULL, object = NULL, control = NULL)
```

```
export_score(keyword = NULL, object = NULL, control = NULL, location = NULL)

export_voi(keyword = NULL, object = NULL, control = NULL)

export_doi(keyword = NULL, object = NULL, control = NULL, locations = NULL)
```

Arguments

control	Integer scalar batch id for control data (batch_c or batch).
location	Character vector (or list coercible via unlist()) of location codes to filter by (e.g., values from countries or us_states).
keyword	Character vector (or list coercible via unlist()) of object keywords to export. When provided, overrides object.
object	Integer scalar batch id for object data (batch_o). Ignored if keyword is supplied.
locations	Character scalar naming a location set (e.g., "countries", "us_states"). Applies to export_doi() only.

Details

All filter arguments default to NULL, which disables that filter and returns all rows for that dimension. When keyword is provided it takes precedence over object: the object argument is silently ignored.

Non-_global functions (export_control(), export_object(), export_score()) exclude the "world" aggregate row. The _global counterparts (export_control_global(), export_object_global(), export_voi()) return only the "world" row.

Value

A data frame with the requested rows and a date column of class Date. Batch identifier columns are renamed for clarity:

- export_control(), export_control_global(): location, keyword, date, hits, control (renamed from batch).
- export_object(), export_object_global(): location, keyword, date, hits, object (from batch_o), control (from batch_c).
- export_score(), export_voi(): location, keyword, date, score, control (from batch_c), object (from batch_o).
- export_doi(): keyword, date, gini, hhi, entropy, control (from batch_c), object (from batch_o), locations.

See Also

- example_control, example_object, example_score, example_doi for the column structure of each table.
- download_control(), download_object() to populate the source tables.
- compute_score(), compute_doi() to compute scores and DOI metrics.
- start_db() to open a database session before exporting.

Examples

```
## Not run:
# Control hits for batch 2
export_control(control = 2)

# World-aggregate control hits
export_control_global(control = 1)

# Object hits for a keyword across all locations
export_object(keyword = "manchester united", location = countries)

# Object hits for multiple keywords
export_object(keyword = c("manchester united", "real madrid"))

# World-aggregate object hits
export_object_global(keyword = "manchester united", control = 1)

# Location-level scores, written to CSV
export_score(object = 3, control = 1, location = us_states) |>
  readr::write_csv("data_score.csv")

# Volume of interest (world-aggregate scores)
export_voi(keyword = "manchester united", control = 1)

# Degree of internationalization for a keyword, written to Excel
export_doi(keyword = "manchester united", control = 2, locations = "us_states") |>
  writexl::write_xlsx("data_doi.xlsx")

## End(Not run)
```

---

get_api_usage *Report daily Research API usage*

Description

Returns the number of Google Trends Research API calls made today, the number remaining before the daily limit is reached, and the limit itself. The counter is stored in `gt.env` and resets automatically when the calendar date changes.

Usage

```
get_api_usage()
```

Details

The counter is incremented once per successful call to the internal helpers `.get_trend()`, `.get_region()`, and `.get_related()` whenever the Research API backend is active (i.e., after `initialize_python()` has been called). Calls routed through the default `gtrendsR` scraping backend are not counted.

The daily limit of 10,000 calls is set by Google. The counter does **not** enforce this limit; it only tracks usage so that callers can monitor their consumption.

Value

A named integer vector with three elements:

`calls` Number of Research API calls made today.

`remaining` Calls remaining before the daily limit is reached.

`limit` The daily limit (always `10000`).

See Also

`initialize_python()` to enable the Research API backend.

Examples

```
get_api_usage()
```

`gt.env` *Package environment for internal state*

Description

`gt.env` is the internal package environment used to store runtime state and data tables. It centralizes objects that should be shared across functions (e.g., data.table stores, cached keyword batches).

Usage

```
gt.env
```

Format

An environment with `parent = emptyenv()`.

Details

The following bindings may be present in `gt.env` after package attach and/or after calling initialization functions such as `start_db()`:

- `dt_keywords`: data.table of keyword batches (type, batch, keyword).
- `dt_time`: data.table of batch time windows (type, batch, start_date, end_date).
- `dt_control`: data.table of control search-volume data.
- `dt_object`: data.table of object search-volume data.
- `dt_score`: data.table of computed scores.
- `dt_doi`: data.table of DOI data.

- dt_locations: data.table of location set definitions.
- dt_region: data.table of regional search-volume data.
- dt_related: data.table of related search terms.
- dt_synonyms: data.table of keyword/synonym mappings.
- keywords_control: Cached data frame of control keywords by batch.
- time_control: Cached data frame of control batch time windows.
- keywords_object: Cached data frame of object keywords by batch.
- time_object: Cached data frame of object batch time windows.
- keyword_synonyms: Cached data frame of keyword/synonym mappings.
- query_wait: Numeric scalar. Seconds to wait between API calls (default: 0.1).
- py_setup: Logical scalar. TRUE if initialize_python() has been called successfully.
- api_calls: Integer scalar. Number of Research API calls made today.
- api_calls_date: Date scalar. The date for which api_calls is counted.
- score_calls: Integer scalar. Number of locations processed by compute_score() since the database was last persisted.

See Also

- start_db()
- initialize_python()

---

initialize_db *Initialize the local database store*

Description

Creates the local database store used by globaltrends in the current working directory and initializes all required tables.

Usage

```
initialize_db()
```

Details

The package uses data.table objects persisted as a single RDS file under the db/ folder. initialize_db() creates 10 empty tables, populates default location sets, and saves the result to db/globaltrends.rds.

If the RDS file already exists the function returns early without overwriting anything.

Default location sets written to data_locations:

countries ISO 3166-1 alpha-2 codes for countries above the GDP share threshold (see countries).

us_states ISO 3166-2 codes for US states and Washington DC (see us_states).

Value

Invisibly returns TRUE. Called for its side effects (creating files under db/).

See Also

start_db() to open a working session after initialization; disconnect_db() to persist changes and close the session.

Examples

```
## Not run:
initialize_db()
start_db()

## End(Not run)
```

---

initialize_python *Initialize Python backend for Google Trends Research API*

Description

Initializes the Python session required to download data via the Google Trends *Research API* (not the public gtrendsR::gtrends() scraping route). The function configures the Python interpreter (Conda or virtualenv), stores the API key in gt.env, sources the package's Python helper code, and marks the session as ready for API-based downloads.

Usage

```
initialize_python(api_key, conda_env = NULL, python_env = NULL)
```

Arguments

api_key	Character scalar. API key obtained from Google.
conda_env	Optional character scalar. Name or path of a Conda environment (passed to reticulate::use_condaenv()). Supply either this or python_env, not both.
python_env	Optional character scalar. Path to a Python virtual environment (passed to reticulate::use_virtualenv Supply either this or conda_env, not both.

Details

Prerequisites. Before calling initialize_python():

1. Apply for Research API access and obtain an API key via Google's request form.
2. Create a Python environment (Conda or virtualenv) with google-api-python-client installed.

Environment specification. Exactly one of `conda_env` or `python_env` must be supplied; providing neither or both is an error.

Effect on the download backend. Once initialized, all download functions (`download_control()`, `download_object()`, `download_region()`, `download_related()`) switch from the default `gtrendsR::gtrends()` scraping route to the Research API.

Value

Invisibly returns `TRUE`. Called for its side effects: stores `api_key` in `gt.env`, sources `python/query_gtrends.py`, and sets `gt.env$py_setup` to `TRUE` to activate the Research API download backend.

See Also

`download_control()`, `download_object()`, `download_region()`, `download_related()` for the download functions that use the Research API once initialized. `reticulate::use_condaenv()` and `reticulate::use_virtualenv()` for Python environment configuration.

Examples

```
## Not run:
# Conda environment
initialize_python(
  api_key   = "YOUR_API_KEY",
  conda_env = "/path/to/conda/env"
)

# Virtual environment
initialize_python(
  api_key    = "YOUR_API_KEY",
  python_env = "/path/to/venv"
)

## End(Not run)
```

---

remove_data *Remove data from database tables*

Description

Removes batches and derived data from the database. Deletions are *greedy*: all downstream tables that depend on the deleted entry are automatically cleaned up to keep the database consistent.

Usage

```
remove_data(table, control = NULL, object = NULL)
```

Arguments

table	Character scalar. The table to delete from. One of "batch_keywords", "batch_time", "data_control", "data_object", "data_score", "data_doi", "data_related", "data_region".
control	Optional integer-like scalar. Control batch id.
object	Optional integer-like scalar. Object batch id.

Details

Dependency chain:

Deletions cascade through the following dependency graph:

```
batch_keywords / batch_time
       |
       v
  data_control
       |
       v
  data_object ---> data_related
       |      \--> data_region
       v
  data_score
       |
       v
   data_doi
```

For example:

- Deleting a control batch from data_control removes all data_object rows for that control, then the associated data_score, data_doi, data_related, and data_region rows.
- Deleting an object batch from batch_keywords removes the corresponding batch_time entry, all data_object rows for that object batch, and everything downstream.

Argument requirements by table:

table	control	object
"batch_keywords", "batch_time"	exactly one of	exactly one of
"data_control"	required	ignored
"data_object", "data_score", "data_doi"	at least one of	at least one of
"data_related", "data_region"	ignored	required

Value

Invisibly returns TRUE on success. The function is called for its side effects (deleting rows).

Examples

```
## Not run:
remove_data(table = "batch_keywords", control = 1)
remove_data(table = "batch_keywords", object = 1)
remove_data(table = "data_object", control = 1)
remove_data(table = "data_score", control = 1, object = 1)
remove_data(table = "data_related", object = 1)
remove_data(table = "data_region", object = 1)

## End(Not run)
```

`start_db` *Start a database session*

Description

Loads the RDS-backed store under `db/` into `data.table` objects in `gt.env`.

Usage

```
start_db()
```

Details

Requires `initialize_db()` to have been run in the current working directory. All tables are read from `db/globaltrends.rds` and assigned into `gt.env` as `dt_*` bindings. Keys are set on the large tables for fast lookups. Cached data frames for frequently-used metadata (`keywords_control`, `keywords_object`, etc.) are also populated. Location sets are exported as named character vectors via `.export_locations()`.

Value

Invisibly returns `TRUE`. Called primarily for its side effects.

See Also

`initialize_db()` to create the store before the first session; `disconnect_db()` to persist changes and close the session.

Examples

```
## Not run:
start_db()
# ... downloads and computations ...
disconnect_db()

## End(Not run)
```

`us_states` *Default location set: US states*

Description

Character vector of US state-level location codes used by the package.

The vector contains the 51 ISO 3166-2 codes of the form `"US-XX"` for the 50 US states and `"US-DC"` for the District of Columbia. Pass this vector as the `locations` argument to `compute_score()` or `compute_doi()` for within-US analyses.

Usage

```
us_states
```

Format

A character vector of 51 ISO 3166-2 location codes.

See Also

`add_locations()`, `start_db()`

Examples

```
length(us_states)
head(us_states)
```

Index

∗ **datasets**
batch_keywords, 8
batch_time, 9
countries, 13
countries_wdi, 14
data_control, 15
data_doi, 16
data_object, 17
data_score, 18
us_states, 36

add_control_keyword, 3
add_keyword *(*add_control_keyword*)*, 3
add_keyword(), *8, 9*, *20–24*, *26, 27*
add_locations, 5
add_locations(), *14*, *36*
add_object_keyword *(*add_control_keyword*)*, 3
add_synonym, 6
add_synonym(), *7, 8*, *12, 13*
aggregate_synonyms, 7
aggregate_synonyms(), *12, 13*

batch_keywords, 8
batch_time, 9

compute_doi, 10
compute_doi(), *6*, *13*, *16, 17*, *28*, *36*
compute_score, 11
compute_score(), *6–8*, *11*, *13*, *17, 18*, *28*, *31*, *36*
compute_voi *(*compute_score*)*, 11
compute_voi(), *12, 13*
countries, *6*, 13, *14*, *28*, *31*
countries_wdi, *13, 14*, 14

data_control, 15
data_doi, 16
data_object, 17
data_score, 18

disconnect_db, 19
disconnect_db(), *32*, *35*
download_control, 19
download_control(), *6*, *13*, *15*, *19, 20*, *23, 24*, *28*, *33*
download_control_global *(*download_control*)*, 19
download_control_global(), *20, 21*
download_object, 21
download_object(), *6*, *11*, *13*, *17*, *21, 22*, *27, 28*, *33*
download_object_global *(*download_object*)*, 21
download_object_global(), *22, 23*
download_region, 23
download_region(), *23, 24*, *33*
download_region_global *(*download_region*)*, 23
download_region_global(), *24*
download_related, 25
download_related(), *33*
download_themes *(*download_related*)*, 25
download_themes(), *25*
download_themes_global *(*download_related*)*, 25
download_themes_global(), *25*
download_themes_rising *(*download_related*)*, 25
download_themes_rising(), *25*
download_themes_rising_global *(*download_related*)*, 25
download_themes_rising_global(), *25*
download_topics *(*download_related*)*, 25
download_topics(), *25*
download_topics_global *(*download_related*)*, 25
download_topics_global(), *25*
download_topics_rising *(*download_related*)*, 25

download_topics_rising(), *25*
download_topics_rising_global
(download_related), 25
download_topics_rising_global(), *25*

example_control, *28*
example_control (data_control), 15
example_doi, *28*
example_doi (data_doi), 16
example_keywords (batch_keywords), 8
example_keywords(), *4*
example_object, *28*
example_object (data_object), 17
example_score, *28*
example_score (data_score), 18
example_time (batch_time), 9
example_time(), *4*
export_control, 27
export_control_global (export_control),
27
export_doi (export_control), 27
export_object (export_control), 27
export_object_global (export_control),
27
export_score (export_control), 27
export_voi (export_control), 27

get_api_usage, 29
gt.env, 30
gtrendsR::countries, *6*

initialize_db, 31
initialize_db(), *19*, *35*
initialize_python, 32
initialize_python(), *20*, *22*, *24*, *26, 27*,
29–31

remove_data, 33
reticulate::use_condaenv(), *32, 33*
reticulate::use_virtualenv(), *32, 33*

start_db, 35
start_db(), *5, 6*, *8, 9*, *12*, *14–24*, *26–28*, *31*,
32, *36*

us_states, *6*, *28*, *31*, 36