Package: metatargetr (via r-universe)

May 28, 2026

Title Retrieve and Parse Meta Ad Targeting Data

Version 0.0.10

Description The metatargetr package provides tools for retrieving, parsing, and analyzing advertising targeting data from Meta's Ad Library and Audience Tab.

License MIT + file LICENSE

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports magrittr, glue, dplyr, purrr, jsonlite, stringr, tibble, httr, rvest, readr, tidyr, httr2, arrow, digest, rlang, xml2, lubridate, progress, cli, janitor, vroom, pacman, chromote (>= 0.2.0)

Suggests testthat (>= 3.0.0), quarto

URL https://github.com/favstats/metatargetr

BugReports https://github.com/favstats/metatargetr/issues

Config/pak/sysreqs chromium cmake git make libicu-dev libxml2-dev libssl-dev libx11-dev

Repository https://favstats.r-universe.dev

Date/Publication 2026-02-27 08:55:08 UTC

RemoteUrl https://github.com/favstats/metatargetr

RemoteRef HEAD

RemoteSha 0b011c21b079031142b04bf7b61c23fdb0596514

Contents

`aggr_targeting` *Aggregate a Pre-Combined Targeting Dataset*

Description

This function takes a single dataframe, assumed to be the result of `bind_rows()` on multiple targeting datasets from different time periods. It correctly aggregates the spending for each unique targeting criterion and calculates the new totals and percentages based on the combined data.

Usage

```
aggr_targeting(combined_df)
```

Arguments

combined_df	A single dataframe that has already been combined from multiple sources (e.g., via dplyr::bind_rows).
filter_disclaimer	An optional character vector of disclaimers or page names to filter the dataset before aggregation. If NULL (default), all data is used.

Value

A single, aggregated tibble where each row represents a unique targeting criterion for an advertiser across the combined period.

---

browser_close *Closes a Playwright browser instance*

Description

This function safely closes the browser instance associated with the provided browser object. It's designed to handle cases where the browser may have already been closed, preventing errors.

Usage

```
browser_close(browser_df)
```

Arguments

browser_df	A tibble returned by browser_launch(), which contains the browser_id of the instance to be closed.

Value

Invisibly returns NULL. This function is called for its side effect of closing the browser.

Examples

```
## Not run:
# Launch a browser
browser <- browser_launch()

# ... perform actions with the browser ...

# Close the browser instance when done
browser_close(browser)

## End(Not run)
```

browser_session_active *Check if a persistent browser session is active*

Description

Verifies both the R-side flag and that Chrome is actually responsive by sending a lightweight evaluation to the browser. If Chrome has crashed or the websocket connection has been lost, the stale session is automatically cleaned up and FALSE is returned.

Usage

```
browser_session_active()
```

Value

Logical, TRUE if a session is active and responsive.

Examples

```
## Not run:
browser_session_active()  # FALSE
browser_session_start()
browser_session_active()  # TRUE
browser_session_close()
browser_session_active()  # FALSE

## End(Not run)
```

browser_session_close *Close the persistent browser session*

Description

Closes the browser session started by `browser_session_start()`. After calling this, each function call will start its own browser again.

Usage

```
browser_session_close()
```

Value

Invisibly returns TRUE on success.

Examples

```
## Not run:
browser_session_start()
# ... do work ...
browser_session_close()

## End(Not run)
```

`browser_session_restart` *Restart the persistent browser session*

Description

Closes the current session (if any) and starts a fresh one with warm-up. Useful when Chrome has become unresponsive or after errors mid-batch.

Usage

```
browser_session_restart(warm_up = TRUE, warm_up_wait = 8)
```

Arguments

`warm_up`	Logical. If TRUE (default), navigates to the Facebook Ad Library landing page on startup to pass the JS challenge and set cookies.
`warm_up_wait`	Seconds to wait during warm-up (default 8).

Value

Invisibly returns TRUE on success.

Examples

```
## Not run:
browser_session_start()
# ... Chrome becomes unresponsive ...
browser_session_restart()
# ... continue working ...
browser_session_close()

## End(Not run)
```

`browser_session_start` *Start a persistent browser session*

Description

Starts a headless Chrome browser session that will be reused by `get_ad_snapshots()`, `get_deeplink()`, and `get_ad_html()` until `browser_session_close()` is called.

Usage

```
browser_session_start(warm_up = TRUE, warm_up_wait = 8)
```

Arguments

`warm_up`	Logical. If TRUE (default), navigates to the Facebook Ad Library landing page on startup to pass the JS challenge and set cookies.
`warm_up_wait`	Seconds to wait during warm-up (default 8).

Details

This significantly improves performance when processing multiple ads, as browser startup (~2–3 seconds) only happens once. By default the session is warmed up by navigating to the Facebook Ad Library landing page, which passes the JS challenge and sets cookies so that subsequent calls return data immediately.

Value

Invisibly returns TRUE on success.

Examples

```
## Not run:
# Start session (includes warm-up)
browser_session_start()

# Process multiple ads (each reuses the session)
results <- map_dfr_progress(ad_ids, ~get_ad_snapshots(.x))

# Close when done
browser_session_close()

## End(Not run)
```

check_hash *Check hash of a media file*

Description

Check hash of a media file

Usage

```
check_hash(.x, hash_table, mediadir)
```

Arguments

.x	Path to the media file.
hash_table	A data frame of existing hashes.
mediadir	Directory to save media files.

Value

A tibble with file details.

create_necessary_dirs *Create necessary directories*

Description

Create necessary directories

Usage

```
create_necessary_dirs(x)
```

Arguments

x	Directory path to check and create.

Value

None (used for side effects).

`detectmysnap` *Detect and parse the snapshot JSON from a Facebook Ad Library script tag*

Description

Facebook's SSR script tags contain a bundle of many ads' snapshot data. This function locates the `"snapshot":` occurrence that belongs to the requested `ad_id` by first finding the ad_id in the raw text and then extracting the nearest `"snapshot":` JSON object after it.

Usage

```
detectmysnap(rawhtmlascharacter, ad_id = NULL)
```

Arguments

`rawhtmlascharacter`	Raw HTML content as character string.
`ad_id`	Character string. The Facebook ad ID to find in the bundle. When provided, the function extracts only the snapshot belonging to this ad. When NULL, extracts the first snapshot found (legacy behavior).

Details

Falls back to the original split-based approach when no ad_id is provided.

Value

A parsed JSON object (list).

See Also

`get_ad_snapshots()` which calls this function internally.

`detectmysnap_dep` *Detect the JSON code on Facebook ad websites*

Description

Function to detect the JSON code on facebook ad websites that contains the media URLs This is basically str_extract but with perl!

Usage

```
detectmysnap_dep(rawhtmlascharacter)
```

Arguments

rawhtmlascharacter	Raw HTML content as character string.

Value

A parsed JSON object.

See Also

detectmysnap

---

download_media *Download media files and potentially hash them*

Description

Download media files and potentially hash them

Usage

```
download_media(media_dat, mediadir = "data/media", hashing = T)
```

Arguments

media_dat	Data containing media URLs.
mediadir	Directory to save media files.
hashing	Logical, whether to hash the files.

Value

None (used for side effects).

---

download_media_int *Download media files with specified IDs*

Description

Download media files with specified IDs

Usage

```
download_media_int(id, x, n, mediadir = "data/media")
```

Arguments

id	Ad ID.
x	Media URLs to download.
n	Number of URLs.
mediadir	Directory to save media files.

Value

A character vector with file paths.

---

extract_media_urls *Extract media URLs from data*

Description

Extract media URLs from data

Usage

```
extract_media_urls(yo)
```

Arguments

yo	Data containing potential media URLs.

Value

A character vector of media URLs.

---

find_name *Find an object in a nested list by name*

Description

Find an object in a nested list by name

Usage

```
find_name(haystack, needle)
```

Arguments

haystack	A nested list.
needle	Name of the object to find.

Value

Object in the nested list with the given name or NULL if not found.

fix_json *Parse JSON-formatted strings into named character vectors*

Description

Parse JSON-formatted strings into named character vectors

Usage

```
fix_json(include)
```

Arguments

include	A character vector of JSON-formatted strings

Value

A list of named character vectors, where each vector represents a parsed JSON object

Examples

```
## Not run:
# Parse an example character vector
example_json <- c("{\"city\":\"Berlin\",\"zip_code\":\"12345\"}",
                  "{\"city\":\"Munich\",\"zip_code\":\"67890\"}")
parsed_json <- fix_json(example_json)

# Check the resulting list of named character vectors
parsed_json

## End(Not run)
```

get_ad_html *Fetch Facebook Ad-Library pages (with caching)*

Description

Retrieves HTML content for Facebook Ad Library pages using headless Chrome to bypass JavaScript-based bot detection. Results are cached to disk.

Usage

```
get_ad_html(
  ad_ids,
  country,
  cache_dir = NULL,
  overwrite = FALSE,
  strip_css = TRUE,
  wait_sec = 3,
  log_failed_ids = NULL,
  quiet = FALSE,
  return_type = c("paths", "list")
)
```

Arguments

ad_ids	Character vector of Ad-Library IDs.
country	Two-letter country code.
cache_dir	Directory where *.html.gz* files will be stored. Defaults to the value set during interactive setup, or "html_cache".
overwrite	If FALSE (default) keep already-cached files.
strip_css	Run fast regex-based CSS removal on downloaded pages.
wait_sec	Seconds to wait for each page to load (default 3).
log_failed_ids	If a character path is provided (e.g., "log.txt"), failed IDs will be appended to that file.
quiet	Suppress progress messages.
return_type	"paths" (default) or "list" for in-memory strings.

Value

Either a named character vector of file paths or a named list of HTML strings, in the *same order* as ad_ids.

---

get_ad_library_page_id

Resolve Facebook Handles to Ad Library Page IDs

Description

Translates Facebook page handles (or profile URLs) into the numeric Ad Library page identifier used as view_all_page_id in https://www.facebook.com/ads/library/ URLs.

Usage

```
get_ad_library_page_id(
  handles,
  wait_sec = 8,
  max_retries = 1,
  country = "ALL",
  quiet = FALSE
)
```

Arguments

handles	Character vector of Facebook handles, profile URLs, profile transparency URLs, Ad Library URLs, or numeric page IDs.
wait_sec	Numeric. Seconds to wait after each page navigation before extracting content. Default is 8.
max_retries	Integer. Number of retries per handle when extraction fails. Default is 1.
country	Character. Country code used when constructing ad_library_url output (default "ALL").
quiet	Logical. If TRUE, suppresses progress messages.

Details

The function loads each page's **Profile Transparency** tab in a headless browser and extracts the Page ID from rendered text.

Value

A tibble with one row per input and columns:

input Original input value.

handle Normalized handle or numeric ID extracted from input.

page_id Resolved Ad Library page ID (same as view_all_page_id).

ad_library_page_id Alias of page_id for clarity.

is_running_ads Logical/NA: whether transparency text indicates the page is currently running ads.

transparency_url Profile transparency URL used for extraction.

ad_library_url Convenience Ad Library page URL for the resolved ID.

ok Logical success flag for each input.

error Error message when resolution fails.

Examples

```
# Numeric IDs are returned directly (no browser needed)
get_ad_library_page_id(c("121264564551002", "106359662726593"))

## Not run:
# Resolve from handles / URLs
```

```
get_ad_library_page_id(c("VVD", "https://www.facebook.com/TeachGoldenApple/"))

## End(Not run)
```

`get_ad_report` *Get Facebook Ad Library Report Data*

Description

Automates downloading Facebook Ad Library reports for vectors of countries, timeframes, and dates. It uses a robust tryCatch block for each request to ensure cleanup and prevent hanging processes.

Usage

```
get_ad_report(country, timeframe, date)
```

Arguments

`country`	A character vector of two-letter ISO country codes.
`timeframe`	A character vector of time windows (e.g., "last_7_days").
`date`	A character vector of report dates in "YYYY-MM-DD" format.

Value

A single tibble containing the combined data for all successful requests.

`get_ad_snapshots` *Get ad snapshots from Facebook Ad Library*

Description

Retrieves snapshot data (images, videos, cards, body text, etc.) for a Facebook ad from the Ad Library. Uses headless Chrome via chromote to bypass Facebook's JavaScript-based bot detection.

Usage

```
get_ad_snapshots(
  ad_id,
  download = FALSE,
  mediadir = "data/media",
  hashing = FALSE,
  wait_sec = 6,
  max_retries = 1
)
```

Arguments

ad_id	Character string. The Facebook ad ID.
download	Logical. If TRUE, download media files to mediadir.
mediadir	Character. Directory to save downloaded media files.
hashing	Logical. If TRUE, hash downloaded files for deduplication. Recommended for large-scale data collection.
wait_sec	Numeric. Seconds to wait for the page to load (default 6). Increase if you are getting empty results.
max_retries	Integer. Number of retry attempts if data is not found on the first try (default 1). Each retry waits progressively longer.

Details

For best results when processing multiple ads, call browser_session_start() before and browser_session_close() after. If no persistent session exists, a temporary one is created and warmed up automatically (adds ~10 seconds on the first call).

Includes built-in retry logic: if the page loads but snapshot data is not yet available (e.g., JS challenge still completing), the function retries with a longer wait before giving up.

Value

A tibble with one row containing ad snapshot fields (images, videos, body, title, display_format, page_name, etc.) and the ad ID. If retrieval fails, returns a single-column tibble with just the id.

Examples

```
## Not run:
# Single ad
snap <- get_ad_snapshots("1536277920797773")

# Batch processing (recommended)
browser_session_start()
results <- map_dfr_progress(ad_ids, ~get_ad_snapshots(.x))
browser_session_close()

## End(Not run)
```

---

get_additional_page_info_db

Get Page Info Dataset for a Specific Country

Description

Downloads the historical Facebook or Instagram page info dataset for a given ISO2C country code. The data is retrieved from a fixed GitHub release URL in `.parquet` format. It includes information on:

- Page-level metadata (e.g., name, verification status, profile type)
- Audience metrics (e.g., number of likes, Instagram followers)
- Shared disclaimers (if applicable)
- Page creation and name change events with timestamps
- Contact and address information (if available)
- Free-text descriptions ("about" section)

Usage

```
get_additional_page_info_db(iso2c, verbose = TRUE)
```

Arguments

`iso2c`	A string specifying the ISO-3166-1 alpha-2 country code (e.g., "DE", "FR", "US").
`verbose`	Logical. If TRUE (default), prints a status message when downloading.

Value

A tibble containing Facebook page info for the specified country. If the dataset is not available or cannot be retrieved, a tibble with `no_data = TRUE` and the given `iso2c` code is returned.

Examples

```
## Not run:
  de_info <- get_page_info_db("DE")
  fr_info <- get_page_info_db("FR")

## End(Not run)
```

`get_ads_info` *Get and Parse Ad Library Data*

Description

A wrapper function that downloads ad HTMLs for a given set of IDs and a country, parses the data, and returns a final, reordered dataframe.

Usage

```
get_ads_info(ad_ids, country, keep_html = TRUE, cache_dir = "html_cache", ...)
```

Arguments

ad_ids	A character vector of Ad Library IDs.
country	A two-letter country code.
keep_html	A logical flag. If FALSE (the default), the cache directory with the downloaded HTML files will be deleted after parsing. If TRUE, the files will be kept.
cache_dir	The directory to store downloaded HTML files. Defaults to "html_cache".
...	Additional arguments to be passed down to get_ad_html() (e.g., overwrite, quiet, max_active).

Value

A single, reordered dataframe containing the parsed ad data.

---

get_deeplink *Extract and flatten 'deeplinkAdCard' JSON from a Facebook Ad Library page*

Description

This function programmatically retrieves the embedded JSON object labeled deeplinkAdCard from the source code of a Facebook Ad Library ad page.

Usage

```
get_deeplink(ad_id, wait_sec = 4)
```

Arguments

ad_id	Character string specifying the Facebook ad ID (as shown in the Ad Library URL).
wait_sec	Seconds to wait for page to load (default 4). Increase if getting errors.

Details

The function performs the following steps internally:

1. Fetches the ad page HTML from Facebook's Ad Library using headless Chrome.
2. Locates the <script> tag containing the deeplinkAdCard object.
3. Uses a recursive regular expression to extract the full JSON object following deeplinkAdCard.
4. Parses the JSON string into a nested R list.
5. Flattens the JSON into a tidy tibble row, unnesting nested sub-objects such as fevInfo, free_form_additional_info, learn_more_content, and optionally snapshot if present.

The output is designed for downstream analysis: each ad is represented as **one row** in a tibble, with nested JSON fields expanded into their own columns via tidyr::unnest_wider().

This function complements get_ad_snapshots(), which extracts the snapshot JSON. Use get_deeplink() when additional metadata embedded under deeplinkAdCard is required.

Value

A tibble with one row, containing flattened columns extracted from the deeplink JSON object. Columns depend on the structure of the JSON and may include fields like `fevInfo_*`, `fevInfo_free_form_additional_int` `fevInfo_learn_more_content_*`, and snapshot-related columns.

See Also

`get_ad_snapshots()` for extracting snapshot JSON; `detectmysnap()` for raw JSON detection.

Examples

```
## Not run:
df <- get_deeplink("1103135646905363")
glimpse(df)

## End(Not run)
```

`get_ggl_ads` *Fetch a Google Ads Transparency Report*

Description

This function automates the process of obtaining data from the Google Ads Transparency report. It targets the main data bundle, which contains several CSV files. The user can specify which file to process using either its full filename or a convenient shorthand. By default, all downloaded files are deleted after the data is read into memory.

Usage

```
get_ggl_ads(file_to_read = "creatives", keep_file_at = NULL, quiet = FALSE)
```

Arguments

`file_to_read`	A character string specifying which CSV file to read from the bundle, using either the full filename or a shorthand alias (e.g., `"creatives"`). Defaults to `"creatives"`.
`keep_file_at`	A character path to a directory where the selected CSV file should be saved. If `NULL` (the default), all downloaded and extracted files are deleted. If a path is provided, the function will save the specified `file_to_read` to that location.
`quiet`	A logical value. If `FALSE` (default), the function will print status messages about downloading and processing.

Details

Downloads the latest Google Political Ads transparency data bundle (a ZIP file), extracts a specific CSV report, reads it into a tibble, and then cleans up the downloaded and extracted files.

The data bundle contains several files. The user can specify which file to read using a shorthand alias.

Available Reports (Aliases):

- `"creatives"` **(Default):** `google-political-ads-creative-stats.csv`. The primary and most detailed file. Contains statistics for each ad creative, including advertiser info, targeting details, and spend.
- `"advertisers"`: `google-political-ads-advertiser-stats.csv`. Aggregate statistics for each political advertiser.
- `"weekly_spend"`: `google-political-ads-advertiser-weekly-spend.csv`. Advertiser spending aggregated by week.
- `"geo_spend"`: `google-political-ads-geo-spend.csv`. Overall spending aggregated by geographic location.
- `"advertiser_geo_spend"`: `google-political-ads-advertiser-geo-spend.csv`. Advertiser-specific spending aggregated by US state.
- `"declarations"`: `google-political-ads-advertiser-declared-stats.csv`. Self-declared information from advertisers in certain regions (e.g., California, New Zealand).
- `"advertiser_mapping"`: `advertiser_id_mapping.csv`. A mapping file to reconcile different advertiser identifiers.
- `"creative_mapping"`: `creative_id_mapping.csv`. A mapping file to reconcile different ad creative identifiers.
- `"updated_date"`: `google-political-ads-updated.csv`. A single-entry file indicating the last time the report data was refreshed.
- `"campaigns"` `(Deprecated)`: `google-political-ads-campaign-targeting.csv`. Ad-level targeting is now in the `"creatives"` file.
- `"keywords"` `(Discontinued)`: `google-political-ads-top-keywords-history.csv`. Historical data on top keywords, terminated in Dec 2019.

For more details on the specific fields in each file, please refer to the Google Ads Transparency Report documentation.

Value

A `tibble` (data frame) containing the data from the selected CSV file.

Examples

```
## Not run:

# Fetch the main creative stats report using the default alias
creative_stats <- get_ggl_ads()

# Fetch the advertiser stats report using its alias
```

```
advertiser_stats <- get_ggl_ads(file_to_read = "advertisers")

# Fetch the advertiser ID mapping file
advertiser_map <- get_ggl_ads(file_to_read = "advertiser_mapping")

# Fetch the geo spend report using its full filename
geo_spend_report <- get_ggl_ads(
  file_to_read = "google-political-ads-geo-spend.csv"
)

# Fetch the main report and save the CSV file to a "data" folder
creative_stats_saved <- get_ggl_ads(
  file_to_read = "creatives",
  keep_file_at = "data/"
)

## End(Not run)
```

`get_linkedin_ads` *Retrieve Ad Data from the LinkedIn Ad Library with Pagination*

Description

This function scrapes ad data from the LinkedIn Ad Library, handling pagination to retrieve all available results for a given search query. It first collects all ad detail links and then scrapes each detail page with a configurable timeout and retry mechanism.

Usage

```
get_linkedin_ads(
  keyword,
  countries,
  start_date,
  end_date,
  account_owner = NULL,
  max_pages = 100,
  max_retries = 5,
  timeout_seconds = 15
)
```

Arguments

`keyword`	A character string for the keyword to search for (e.g., "Habeck").
`countries`	A character vector of two-letter country codes (e.g., "DE").
`start_date`	The start date of the search range in "YYYY-MM-DD" format.
`end_date`	The end date of the search range in "YYYY-MM-DD" format.
`account_owner`	Optional. A character string for the ad account owner.

max_pages The maximum number of pages to scrape. Defaults to 100.

max_retries The maximum number of retries for each detail page request. Defaults to 5.

timeout_seconds
The timeout in seconds for each detail page request. Defaults to 15.

Value

A tibble containing the detailed scraped ad information from all pages.

Examples

```
## Not run:
  ads_data <- get_linkedin_ads(
    keyword = "Habeck",
    countries = "DE",
    start_date = "2025-01-01",
    end_date = "2025-02-23",
    account_owner = "INSM",
    max_pages = 5,
    max_retries = 3,
    timeout_seconds = 20
  )
  print(ads_data)

## End(Not run)
```

---

get_page_info_db *Get Page Info Dataset for a Specific Country*

Description

Downloads the historical Facebook or Instagram page info dataset for a given ISO2C country code. The data is retrieved from a fixed GitHub release URL in `.parquet` format. It includes information on:

- Page-level metadata (e.g., name, verification status, profile type)
- Audience metrics (e.g., number of likes, Instagram followers)
- Shared disclaimers (if applicable)
- Page creation and name change events with timestamps
- Contact and address information (if available)
- Free-text descriptions ("about" section)

Usage

```
get_page_info_db(iso2c, verbose = TRUE)
```

Arguments

iso2c	A string specifying the ISO-3166-1 alpha-2 country code (e.g., "DE", "FR", "US").
verbose	Logical. If TRUE (default), prints a status message when downloading.

Value

A tibble containing Facebook page info for the specified country. If the dataset is not available or cannot be retrieved, a tibble with `no_data = TRUE` and the given `iso2c` code is returned.

Examples

```
## Not run:
  de_info <- get_page_info_db("DE")
  fr_info <- get_page_info_db("FR")

## End(Not run)
```

---

get_page_insights *Get Page Insights*

Description

Retrieves insights for a given Facebook page within a specified timeframe, language, and country. It allows for fetching specific types of information and optionally joining page info with targeting info.

Usage

```
get_page_insights(
  pageid,
  timeframe = "LAST_30_DAYS",
  lang = "en-GB",
  iso2c = "US",
  include_info = c("page_info", "targeting_info"),
  join_info = T
)
```

Arguments

pageid	A string specifying the unique identifier of the Facebook page.
timeframe	A string indicating the timeframe for the insights. Valid options include predefined timeframes such as "LAST_30_DAYS". The default value is "LAST_30_DAYS".
lang	A string representing the language locale to use for the request, formatted as language code followed by country code (e.g., "en-GB" for English, United Kingdom). The default is "en-GB".

iso2c	A string specifying the ISO-3166-1 alpha-2 country code for which insights are requested. The default is "US".
include_info	A character vector specifying the types of information to include in the output. Possible values are "page_info" and "targeting_info". By default, both types of information are included.
join_info	A logical value indicating whether to join page info and targeting info into a single data frame (if TRUE) or return them as separate elements in a list (if FALSE). The default is TRUE.

Value

If join_info is TRUE, returns a data frame combining page and targeting information for the specified Facebook page. If join_info is FALSE, returns a list with two elements: page_info and targeting_info, each containing the respective data as a data frame. In case of errors or no data available, the function may return a simplified data frame or list indicating the absence of data.

Examples

```
insights <- get_page_insights(pageid="123456789", timeframe="LAST_30_DAYS", lang="en-GB", iso2c="US",
                              include_info=c("page_info", "targeting_info"), join_info=TRUE)
```

---

get_report_db *Retrieve Report Data from GitHub Repository*

Description

This function retrieves a report for a specific country and timeframe from a GitHub repository hosting RDS files. The file is downloaded to a temporary location, read into R, and then deleted.

Usage

```
get_report_db(the_cntry, timeframe, ds, verbose = FALSE)
```

Arguments

the_cntry	Character. The ISO country code (e.g., "DE", "US").
timeframe	Character or Numeric. Timeframe in days (e.g., "30", "90") or "yesterday" / "lifelong".
ds	Character. A timestamp or identifier used to construct the file name (e.g., "2024-12-25").
verbose	Logical. Whether to print messages about the process. Default is FALSE.

Value

A data frame or object read from the RDS file.

Examples

```
# Example usage
report_data <- get_report_db(
  the_cntry = "DE",
  timeframe = 30,
  ds = "2024-12-25",
  verbose = TRUE
)
print(head(report_data))
```

`get_targeting` *Get Meta Ad Library targeting data for a page*

Description

This function retrieves data for the targeting criteria of a Facebook page for a specified timeframe.

Usage

```
get_targeting(id, timeframe = "LAST_30_DAYS", lang = "en-GB", legacy = FALSE)
```

Arguments

`id`	A character string representing the Facebook page ID.
`timeframe`	A character string representing the desired timeframe. Can either be "LAST_30_DAYS" or "LAST_7_DAYS". Defaults to "LAST_30_DAYS".
`lang`	An ISO language code character string representing the desired language of the targeting criteria. Defaults to "en-GB" but can be "en-US" and many more.

Value

A `tibble` containing the audience data for the specified Facebook page and timeframe.

Examples

```
## Not run:
get_targeting("123456789")
get_targeting("987654321", "LAST_7_DAYS")

## End(Not run)
```

`get_targeting_db` *Retrieve Targeting Data from GitHub Repository*

Description

This function retrieves targeting data for a specific country and timeframe from a GitHub repository hosting parquet files. The function uses the `arrow` package to read the parquet file directly from the specified URL. Note that the retreival of archived data is only possible three days after a specified date.

Usage

```
get_targeting_db(the_cntry, tf, ds, remove_nas = T, verbose = F)
```

Arguments

`the_cntry`	Character. The ISO country code (e.g., "DE", "US").
`tf`	Numeric or character. The timeframe in days ("yesterday", "7", "30", "90", "lifelong"). Note, some data points for lifelong in the past may be missing for some countries.
`ds`	Character. A timestamp or identifier used to construct the file path (e.g., "2024-12-25").

Value

A data frame containing the targeting data from the parquet file.

Examples

```
# Example usage
latest_data <- get_targeting_db(
  the_cntry = "DE",
  tf = 30,
  ds = "2024-10-25"
)
print(head(latest_data))
```

`get_targeting_metadata`

Retrieve Metadata for Targeting Data

Description

This function retrieves metadata for targeting data releases for a specific country and timeframe from a GitHub repository.

Usage

```
get_targeting_metadata(
  country_code,
  timeframe,
 base_url = "https://github.com/favstats/meta_ad_targeting/releases/expanded_assets/"
)
```

Arguments

country_code	Character. The ISO country code (e.g., "DE", "US").
timeframe	Character. The timeframe to filter (e.g., "7", "30", or "90").
base_url	Character. The base URL for the GitHub repository. Defaults to "https://github.com/favstats/meta

Value

A data frame containing metadata about available targeting data, including file names, sizes, timestamps, and tags.

Examples

```
# Retrieve metadata for Germany for the last 30 days
metadata <- get_targeting_metadata("DE", "30")
print(metadata)
```

---

ggl_get_spending *Retrieve Google Ad Spending Data*

Description

This function queries the Google Ads Transparency Report to retrieve information about advertising spending for a specified advertiser. It supports a range of countries and can return either aggregated data or time-based spending data.

Usage

```
ggl_get_spending(
  advertiser_id,
  start_date = 20231029,
  end_date = 20231128,
  cntry = "NL",
  get_times = FALSE
)
```

Arguments

advertiser_id	A string representing the unique identifier of the advertiser. For example "AR14716708051084115969".
start_date	An integer representing the start date for data retrieval in YYYYMMDD format. For example 20231029.
end_date	An integer representing the end date for data retrieval in YYYYMMDD format. For example 20231128.
cntry	A string representing the country code for which the data is to be retrieved. For example "NL" (Netherlands).
get_times	A boolean indicating whether to return time-based spending data. If FALSE, returns aggregated data. Default is FALSE.

Value

A tibble containing advertising spending data. If `get_times` is TRUE, the function returns a tibble with date-wise spending data. Otherwise, it returns a tibble with aggregated spending data, including details like currency, number of ads, ad type breakdown, advertiser details, and other metrics.

Examples

```
# Retrieve aggregated spending data for a specific advertiser in the Netherlands
spending_data <- ggl_get_spending(advertiser_id = "AR14716708051084115969",
                                  start_date = 20231029, end_date = 20231128,
                                  cntry = "NL")

# Retrieve time-based spending data for the same advertiser and country
time_based_data <- ggl_get_spending(advertiser_id = "AR14716708051084115969",
                                    start_date = 20231029, end_date = 20231128,
                                    cntry = "NL", get_times = TRUE)
```

---

map_dfr_progress *Apply a function to each element of a list with a progress bar*

Description

Adding a progress bar to the map_dfr function https://www.jamesatkins.net/posts/progress-bar-in-purrr-map-df/

Usage

```
map_dfr_progress(.x, .f, ...)
```

Arguments

.x	List to iterate over.
.f	Function to apply.
...	Other parameters passed to `purrr::map_dfr`.
.id	An identifier.

Value

An aggregated data frame.

`parse_linkedin_ads_details` *Parse Important Information from an Ad Detail HTML Page*

Description

This function takes the HTML content from a LinkedIn Ad Library detail page and extracts key information like advertiser, targeting, and impressions.

Usage

```
parse_linkedin_ads_details(html_content)
```

Arguments

`html_content`	An `xml_document` object, typically read from an HTML file or obtained from an HTTP response.

Value

A list containing the extracted ad details.

Examples

```
## Not run:
  # Assuming you have saved the HTML of a detail page to "ad_detail.html"
  ad_html <- rvest::read_html("ad_detail.html")
  details <- parse_ad_details(ad_html)
  print(details)

## End(Not run)
```

`parse_location` *Parse Location from Ad Targeting Dataset*

Description

A function to parse the location strings in the Ad Targeting Dataset and split into separate columns for each level of detail.

Usage

```
parse_location(.x, loc_var, type = "include", verbose = T)
```

Arguments

.x	A data.frame containing the location string
loc_var	A character string specifying the name of the column in .x that contains the location string
type	A character string specifying the prefix to add to each column of split location details. Default is "include". Should be "include" or "exclude".
verbose	A logical flag specifying whether to display a progress bar during processing. Default is TRUE.

Value

A data.frame with columns for each level of detail in the location.

Examples

```
## Not run:
  ### create a dataset with unique include_location values
  distinct_data <- targeting_data %>%
    distinct(include_location)

  #### parse the location data and join in original dataset
  distinct_data %>%
    parse_location(include_location, type = "include") %>%
    right_join(targeting_data)

  ###----####

  ### create a dataset with unique exclude_location values
  distinct_data <- targeting_data %>%
    distinct(exclude_location)

  #### parse the location data and join in original dataset
  distinct_data %>%
    parse_location(exclude_location, type = "exclude") %>%
    right_join(targeting_data)


## End(Not run)
```

---

search_ad_library *Search the Facebook Ad Library*

Description

Retrieves ads from the Facebook Ad Library by page ID or text query, **without requiring an API key**. Uses headless Chrome to load the Ad Library search page, which embeds ~30 ads with rich metadata in its server-side rendered HTML.

Usage

```
search_ad_library(
  page_id = NULL,
  query = NULL,
  country = "ALL",
  ad_type = "all",
  active_status = "all",
  date_min = NULL,
  date_max = NULL,
  media_type = "all",
  publisher_platforms = NULL,
  content_languages = NULL,
  search_type = NULL,
  sort_mode = "total_impressions",
  sort_direction = "desc",
  max_pages = 1,
  wait_sec = 12
)
```

Arguments

page_id	Character. Facebook page ID to search for all ads from that page (e.g., "52985377549" for D66). Use this for targeted searches of a specific advertiser. Mutually exclusive with query.
query	Character. Text search query (e.g., "Rob Jetten", "coca-cola"). Returns ads mentioning the query from any advertiser. Mutually exclusive with page_id.
country	Character. ISO country code filter (default "ALL"). Use "NL" for Netherlands, "US" for United States, etc.
ad_type	Character. Ad type filter (default "all"). Options: "all", "political_and_issue_ads".
active_status	Character. Delivery status filter (default "all"). Options: "all", "active", "inactive".
date_min	Character or NULL. Minimum start date filter in YYYY-MM-DD format. Maps to start_date[min] in the Ad Library URL.
date_max	Character or NULL. Maximum start date filter in YYYY-MM-DD format. Maps to start_date[max] in the Ad Library URL.
media_type	Character. Media type filter (default "all"). Common options: "all", "image", "video".
publisher_platforms	Character vector or NULL. Platform filters. Example: c("facebook", "instagram").
content_languages	Character vector or NULL. Content language filters. Example: c("nl", "en").
search_type	Character or NULL. Search type override. Common options: "keyword_unordered", "keyword_exact_phrase", "page". If NULL (default), inferred from page_id/query.
sort_mode	Character or NULL. Sort mode for sort_data[mode]. Default "total_impressions".
sort_direction	Character or NULL. Sort direction for sort_data[direction]. Default "desc".

max_pages	Integer. Number of pages to fetch, each containing ~30 ads (default 1). Set higher to paginate (experimental).
wait_sec	Numeric. Seconds to wait for the page to load (default 12). Increase if getting empty results.

Details

Data returned:

Each ad includes:

- **Metadata**: ad_archive_id, page_name, page_id, categories (e.g., "POLITICAL"), is_active, start_date, end_date, spend, currency, reach_estimate, publisher_platform, impressions_lower, impressions_upper, targeted_countries
- **Creative (snapshot)**: body, title, display_format, link_url, cta_text, images, videos, cards, page_profile_picture_url
- **Link**: ad_library_url — direct URL to the ad in Facebook's Ad Library

Important notes:

- Uses active_status="all" by default (full history)
- Use active_status="active" to focus on ads currently running
- Each page load yields ~30 ads from the server-side rendered HTML
- Pagination beyond page 1 reuses Facebook's own pagination request shape captured from browser runtime (still experimental)
- The categories field can identify political ads ("POLITICAL")

Value

A tibble with one row per ad. Returns an empty tibble if no ads are found. See **Data returned** section for column details.

Examples

```
## Not run:
browser_session_start()

# Search by page ID (all D66 ads)
d66_ads <- search_ad_library(page_id = "52985377549")

# Search by keyword in the Netherlands
ads <- search_ad_library(query = "Rob Jetten", country = "NL")

# Check which ads are political
ads %>% dplyr::filter(categories == "POLITICAL")

# Check ad dates and spend
ads %>% dplyr::select(page_name, start_date, end_date, spend, categories)

browser_session_close()

## End(Not run)
```

set_metatargetr_options *Interactively Set and Save User Settings*

Description

Launches an interactive command-line interface to help users configure and save default package options, such as the cache directory and user-agent randomization preference.

Usage

```
set_metatargetr_options()
```

Details

The function will guide the user through setting the following options:

- `metatargetr.cache_dir`: The default directory to save HTML files.
- `metatargetr.randomize_ua`: Whether to use random User-Agents by default.

The user will also be prompted to save these settings as environment variables in their personal `.Renviron` file for persistence across R sessions.

stupid_conversion *Convert an entry for an ad into a row in a tibble*

Description

We did not manage to easily output the entry for one ad as a row in a tibble; this line solves the issue and does exactly that.

Usage

```
stupid_conversion(x)
```

Arguments

x	An object containing data about the ad.

Value

A tibble row.

unnest_and_fix_dups *Unnest interest targeting data*

Description

unnest and fix duplicates in interest targeting data from Ad Targeting Dataset

The function unnests "include" and "exclude" columns in the Ad Targeting Dataset, and removes duplicates.

Usage

```
unnest_and_fix_dups(dat, the_list, new_name)
```

Arguments

dat	a data frame
the_list	the column name to unnest
new_name	the name of the new column after unnesting and fixing duplicates

Value

a modified data frame with unnested and deduplicated values

Examples

```
### example usage:
## make sure you have the variable 'archive_id' in your data
ad_targeting_data %>%
    rowwise() %>%
    mutate(include_list = fix_json(include)) %>%
    ungroup() %>%
   ## the_list: the parsed list of JSON, new_name: what the parsed column should be called
   unnest_and_fix_dups(the_list = include_list, new_name = "parsed_include")
```

walk_progress *Walk through a list with a progress bar*

Description

Walk through a list with a progress bar

Usage

```
walk_progress(.x, .f, ...)
```

Arguments

.x	List to iterate over.
.f	Function to apply.
...	Other parameters passed to purrr::map_dfr.

Value

None (used for side effects).

Index