

Package: spacescans (via r-universe)

October 28, 2026

Type Package

Title Link Time-Referenced Geolocation Data to the Spatial and Contextual Exposome

Version 0.3.4

Description R client for the 'spacescans' 'Python' pipeline, which links time-referenced geolocation data (location episodes of study participants) to the spatial and contextual exposome and returns one row per participant or per location episode. The pipeline runs in a separate 'Python' process that the package talks to through JSON request and result files and 'Parquet' tables, so a crash or an interrupt in 'Python' never takes the R session down. 'ss_install()' builds the 'Python' environment with 'uv'.

License MIT + file LICENSE

URL <https://github.com/IU-Ultraman/spacescans>

BugReports <https://github.com/IU-Ultraman/spacescans/issues>

Depends R (>= 4.1.0)

Imports jsonlite, nanoparquet, processx, ps, stats, tools, utils

Suggests arrow, bit64, data.table, knitr, markdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

SystemRequirements Python 3.12 with the 'spacescans' Python package, installed by ss_install() with 'uv' (<<https://docs.astral.sh/uv/>>); macOS 14 or later on Apple silicon is the supported platform.

Config/spacescans/version 0.3.4

Config/spacescans/protocol 1

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1
Config/pak/sysreqs python3
Repository <https://iu-ultraman.r-universe.dev>
Date/Publication 2026-10-27 20:30:38 UTC
RemoteUrl <https://github.com/IU-Ultraman/spacescans>
RemoteRef v0.3.4
RemoteSha b49626c8c45863601fdb5e3a4c4a98af8fd07238
RemoteSubdir r

Contents

spacescans-conditions	3
ss_cache_info	4
ss_catalog	5
ss_config_schema	5
ss_coverage	6
ss_custom_polygon	7
ss_data_dir	8
ss_data_status	9
ss_data_verify	10
ss_datasets	11
ss_doctor	14
ss_features	15
ss_fetch	15
ss_install	16
ss_job	18
ss_link	20
ss_quickstart	23
ss_read_config	24
ss_read_result	25
ss_run	25
ss_templates	27
ss_use_python	27
ss_validate	28
ss_version	29
Index	30

spacescans-conditions	<i>Conditions raised by spacescans</i>
-----------------------	--

Description

Every error the package raises inherits from `spacescans_error`, plus a class for its kind, so callers can catch exactly what they expect:

Details

kind	class	when
protocol	<code>spacescans_error_protocol</code>	versions or protocol do not match
python	<code>spacescans_error_python</code>	no usable Python interpreter
install	<code>spacescans_error_install</code>	<code>ss_install()</code> failed
config	<code>spacescans_error_config</code>	invalid arguments, unknown variable
cohort	<code>spacescans_error_cohort</code>	a required column is missing, a data file
data	<code>spacescans_error_data</code>	an input file is missing or unreadable
data_missing	<code>spacescans_error_data_missing</code> (also <code>spacescans_error_data</code>)	the data a variable needs is not in the file
data_corrupt	<code>spacescans_error_data_corrupt</code> (also <code>spacescans_error_data</code>)	a checksum or size does not match
terms_required	<code>spacescans_error_terms</code>	a provider's terms must be accepted
missing_extra	<code>spacescans_error_missing_extra</code>	a pattern needs a Python extra that is not installed
network	<code>spacescans_error_network</code>	a download failed after retries; run again
disk_space	<code>spacescans_error_disk</code>	not enough free space for a download
locked	<code>spacescans_error_locked</code>	another download holds the data file
step_failed	<code>spacescans_error_step</code>	a pipeline step exited with a non-zero status
runtime	<code>spacescans_error_runtime</code>	the pipeline raised an error
interrupted	<code>spacescans_error_interrupted</code>	Ctrl-C, SIGINT or SIGTERM
crashed	<code>spacescans_error_crashed</code>	the Python process died without a message
internal	<code>spacescans_error_internal</code>	a bug in the bridge or the package

Kinds added later by the bridge map to `spacescans_error_<kind>` automatically.

The condition object carries `kind`, `hint`, `details` (from Python), `py_type` and `py_traceback` (the Python exception), `stderr` (the last lines of the bridge log), `exit_status` and `run_dir` where they apply.

Examples

```
tryCatch(  
  spacescans:::ss_abort("config", "bad value", hint = "fix it"),  
  spacescans_error_config = function(e) conditionMessage(e)  
)
```

ss_cache_info

*The C3 weight cache***Description**

`ss_link()` keeps the C3 weights it computes (which census units, grid cells or features each location's buffer overlaps) in the user cache directory, so linking the same cohort with the same buffer again skips the slowest steps. Each entry is keyed by the cohort's coordinates, the buffer and raster settings, the code and data fingerprints and the native library versions; a change in any of them is a miss. The entries can reveal where cohort members lived, so the directory is private (0700) and you may want to clear it when a project ends.

Usage

```
ss_cache_info(cache_dir = NULL)
```

```
ss_cache_clear(older_than = NULL, cache_dir = NULL)
```

Arguments

cache_dir	The cache root (default: option <code>spacescans.cache_dir</code> , then <code>tools::R_user_dir("spacescans", "cache")</code>).
older_than	Remove only entries older than this: a number of days or a difftime . NULL (default) removes every entry.

Details

Both functions also sweep the run directories under `tools::R_user_dir("spacescans", "cache")/runs`: a directory whose owner process no longer exists (an R session that was killed) still holds that run's cohort file and outputs, so it is deleted, never archived, and a message says how many went. The same sweep runs when the package loads and in `ss_doctor()`.

Value

`ss_cache_info()`: a list with `cache_dir`, `entries`, `bytes`, `oldest`, `newest`, `temp_files/temp_bytes` (copies left by writes that were interrupted; they hold weights too and `ss_cache_clear()` removes them) and `runs` (the run-directory sweep: removed, kept, unknown, bytes). `ss_cache_clear()`: invisibly, what was removed.

Examples

```
## Not run:
ss_cache_info()
ss_cache_clear(older_than = 90)

## End(Not run)
```

ss_catalog

The exposure catalog

Description

Lists the catalog variables: label, data source, years, spatial unit, output columns, the datasets each needs and the Python extras its pattern needs. With data_dir, a status column says whether the data is in place (see [ss_data_status\(\)](#)).

Usage

```
ss_catalog(data_dir = NULL)
```

Arguments

data_dir A data directory to check, or NULL (default) for the catalog alone.

Value

A data frame with one row per variable (variable, label, experiment, data_source, year_min, year_max, boundary, value_cols, datasets, extras, ...; list-valued fields are joined with ", "), plus status and usable when data_dir is given.

Examples

```
## Not run:
cat <- ss_catalog()
cat[, c("variable", "label", "year_min", "year_max")]

## End(Not run)
```

ss_config_schema

The JSON schema of a pipeline config

Description

The schema of DatasetConfig, the model every C3 and C4 config is validated against (fields, types, defaults and allowed values).

Usage

```
ss_config_schema(as = c("list", "json"))
```

Arguments

as "list" (default) for a nested R list, or "json" for the schema as a JSON string.

Value

The schema.

Examples

```
## Not run:
schema <- ss_config_schema()
names(schema$properties)

## End(Not run)
```

ss_coverage	<i>Estimate coverage before linking</i>
-------------	---

Description

For each variable, how many location episodes of the cohort fall inside its years and region, from the catalog’s coverage years and a bounding box for the contiguous US. It runs no pipeline step and needs no data. Like [ss_link\(\)](#), only the five required columns are handed to Python. The estimate counts location episodes whatever output level you link at: it checks dates and locations, before any value exists.

Usage

```
ss_coverage(cohort, variables, cols = NULL, custom = NULL)
```

Arguments

cohort	A data frame with one row per location episode, or the path of a CSV or Parquet file.
variables	Catalog variable keys, for example <code>c("ndi", "walkability")</code> (see ss_catalog()). The variables of custom datasets are added automatically, so <code>character()</code> links the custom datasets alone.
cols	Your column names for the required columns, for example <code>c(pid = "participant_id", startDate = "start", endDate = "end", longitude = "lon", latitude = "lat")</code> .
custom	Your own datasets: a dataset registered with ss_register_dataset() (its name, id or variable key, or the record), a definition from ss_custom_polygon() or ss_custom_raster() , or a list (or character vector) of them, linked in that order.

Value

A data frame with one row per variable: variable, coverage_pct (or pct), episodes in the time window, in the region, covered, and notes; attribute episodes, the number of location episodes.

Examples

```
## Not run:
ss_coverage(cohort, c("ndi", "temis"))

## End(Not run)
```

ss_custom_polygon	<i>Bring your own exposure data</i>
-------------------	-------------------------------------

Description

Validates your own values so [ss_link\(\)](#) can link them like a catalog variable:

- `ss_custom_polygon()`: a CSV (or data frame) of values keyed by a Census geography (Tract, BG, ZCTA5 or County), static or one row per year;
- `ss_custom_raster()`: a GeoTIFF, or one per year on the same grid, with one or more bands (needs the geo extra).

Usage

```
ss_custom_polygon(
  values,
  boundary = c("Tract", "BG", "ZCTA5", "County"),
  key_col,
  value_cols,
  year_col = NULL,
  name = NULL,
  labels = NULL,
  units = NULL,
  description = NULL
)

ss_custom_raster(
  paths,
  years = NULL,
  bands = 1L,
  value_cols = NULL,
  name = NULL,
  labels = NULL,
  units = NULL,
  description = NULL
)
```

Arguments

values	Path of a CSV file, or a data frame (written to a CSV file of its own under tempdir() for this session).
--------	--

boundary	The geography the key column holds.
key_col	Column with the geography id (GEOID).
value_cols	Columns with the values to link.
year_col	Column with the year, for yearly data; NULL for static.
name	A label for the dataset (default: the file name).
labels, units	Named character vectors of labels and units, keyed by value column.
description	A sentence describing the dataset (default: one made from the boundary or the raster).
paths	GeoTIFF paths.
years	The year of each raster, or NULL for one static raster.
bands	Band numbers to read.

Details

The definition points at your files where they are; keep them in place until the link has run. Pass it to `ss_link(custom = )`; its key (`custom_id`) is added to the variables automatically. To prepare a dataset once and link it by name in later sessions (also daily tables, biweekly or daily rasters, and point, line or polygon features), register it with `ss_register_dataset()` instead.

Value

A definition: a list of class `ss_custom` (`variable_key`, `label`, `boundary`, `value_cols`, ...). The `features` attribute of the `ss_link()` result describes each of its columns from this definition: the column's label (else the dataset's name), and the description followed by the column's unit.

Examples

```
## Not run:
svi <- ss_custom_polygon("svi_tract.csv", boundary = "Tract",
                        key_col = "FIPS", value_cols = c("RPL_THEMES"))
res <- ss_link(cohort, "ndi", custom = svi)

## End(Not run)
```

ss_data_dir

The data directory

Description

Reads or sets the directory that holds the exposure data. There is no implicit default: it is what `ss_data_dir(path)` set in this session (option `spacescans.data_dir`), otherwise the environment variable `SPACESCANS_DATA_DIR`. No data ships with `spacescans`; `ss_fetch()` downloads it into this directory.

Usage

```
ss_data_dir(path = NULL)
```

Arguments

path A directory (it may not exist yet). NULL reads the current setting.

Value

The absolute path (invisibly when setting). Reading raises an error of class `spacescans_error_config` when nothing is set.

Examples

```
old <- getOption("spacescans.data_dir")
ss_data_dir(file.path(tempdir(), "ss-data"))
ss_data_dir()
options(spacescans.data_dir = old)
```

ss_data_status	<i>What is in the data directory</i>
----------------	--------------------------------------

Description

For each variable, whether the data it needs is in place and what to do next. Statuses: `ready`, `differs_from_reference` and `unverified` can be used; `incomplete`, `missing`, `action_required` (a provider download by hand, terms, or a file not yet on the mirror) and `corrupt` cannot. Nothing is written.

Usage

```
ss_data_status(
  variables = NULL,
  data_dir = ss_data_dir(),
  deep = FALSE,
  datasets = NULL
)
```

Arguments

variables Variable keys; NULL checks every catalog variable.

data_dir The data directory (default `ss_data_dir()`).

deep Also hash every file with a recorded checksum (reads the whole tree; slow).

datasets Dataset ids to check as well (or instead).

Value

A data frame of class `ss_data_status` with one row per variable (variable, status, usable, datasets, download_bytes, disk_bytes, next), and attributes datasets (one row per dataset), ready, next_steps, warnings, data_dir and data_release.

Examples

```
## Not run:
ss_data_status(c("ndi", "walkability"))

## End(Not run)
```

ss_data_verify

Verify, import and read terms of datasets

Description

- `ss_data_verify()` checks the files in the data directory against the manifest and their receipts (sizes; with `deep = TRUE` also SHA-256).
- `ss_data_import()` places files you downloaded by hand (for example VNL or ACAG from the provider) into the data directory and records receipts.
- `ss_data_terms()` shows a dataset's licence, attribution, terms and download instructions.

Usage

```
ss_data_verify(
  variables = NULL,
  datasets = NULL,
  data_dir = ss_data_dir(),
  deep = FALSE
)
```

```
ss_data_import(dataset, paths, data_dir = ss_data_dir(), move = FALSE)
```

```
ss_data_terms(dataset)
```

Arguments

variables	Variable keys; NULL checks every catalog variable.
datasets	Dataset ids to check as well (or instead).
data_dir	The data directory (default <code>ss_data_dir()</code>).
deep	Also hash every file with a recorded checksum (reads the whole tree; slow).
dataset	A dataset id (see the datasets attribute of <code>ss_data_status()</code>).
paths	Files or directories to import.
move	Move instead of copy.

Value

The bridge's report as a list (ss_data_terms()): class ss_terms, printed).

Examples

```
## Not run:
ss_data_verify(c("ndi"), deep = TRUE)
ss_data_terms("faqsd")
ss_data_import("vn1", "~/Downloads/vn1")

## End(Not run)
```

ss_datasets

Registered datasets: your own data, prepared once and reused

Description

Registers your own exposure data once, so that later [ss_link\(\)](#) calls use it by name like a catalog variable: ss_link(cohort, "ndi", custom = "my dataset").

Usage

```
ss_register_dataset(
  paths,
  name,
  kind,
  temporal,
  key = NULL,
  variables = NULL,
  periods = NULL,
  description = NULL,
  source = NULL,
  version = NULL,
  license_note = NULL,
  visibility = c("private", "deployment"),
  owner = NULL,
  root = NULL,
  overwrite = FALSE
)

ss_datasets(root = NULL)

ss_dataset(name_or_id, root = NULL)

ss_update_dataset(
  name_or_id,
  name = NULL,
```

```

    description = NULL,
    source = NULL,
    version = NULL,
    license_note = NULL,
    visibility = NULL,
    owner = NULL,
    variables = NULL,
    root = NULL
)

ss_remove_dataset(name_or_id, root = NULL)

```

Arguments

paths	The data files: one CSV for a table; GeoTIFF files for a raster; a GeoPackage, GeoJSON or zipped Shapefile per year (or one file) for features. For a polygon table, a data frame is also accepted (written to a private temporary CSV file first).
name	A name for the dataset, unique in the registry (1 to 80 characters), for example "SVI 2020". For <code>ss_update_dataset()</code> , the new name.
kind	"polygon_table", "raster" or "features".
temporal	"static", "yearly", "biweekly" (rasters) or "daily" (tables and rasters); features are "static" or "yearly".
key	How places (and times) are read from a polygon table: a geography name or a named list (see Details); optional for features (<code>list(year_column = , layer =)</code>); unused for rasters.
variables	The variables: a character vector of names, or a data frame with a name column and optionally unit, description, and column (a table's column, when it differs from the name) or band (a raster band). NULL takes the defaults: every numeric column of a table; band 1 of a raster, named after the dataset; <code><name>_dist_m</code> for features. For <code>ss_update_dataset()</code> , a data frame with name and the new unit and/or description of existing variables.
periods	For rasters and yearly features: the year (a number) or the start date (a Date or "2016-01-01") of each file, in the order of paths, when the file names do not carry them.
description, source, version, license_note	Optional metadata: a description of the dataset, where it comes from, its version, and the terms under which it may be used.
visibility	"private" (default) or "deployment".
owner	Optional owner (any text, for example your name).
root	The registry directory (see Details).
overwrite	Replace a dataset of the same name (it gets a new id).
name_or_id	A dataset's name, id or variable key, or the record from <code>ss_register_dataset()</code> or <code>ss_dataset()</code> .

Details

- `ss_register_dataset()` checks the files and prepares them once (normalized Parquet for tables; checked copies and a period index for rasters; GeoParquet in EPSG:5070 for features). The prepared files and a `dataset.json` record go to the registry; your original files are not changed and are no longer needed by the link.
- `ss_datasets()` lists the registered datasets.
- `ss_dataset()` returns one, with its variables and prepared files.
- `ss_update_dataset()` changes a dataset's name, metadata (including visibility) or the units and descriptions of its variables; the data stays.
- `ss_remove_dataset()` removes one: its record and prepared files (not your original files).

The work is done by the registry of the Python package (`spacescans.custom.register()` and its companions) through the bridge, so a dataset registered from Python is visible from R and the other way round. A dataset is found by its name (ignoring case), its id or its variable key (`custom_` and the first 8 characters of the id).

Kinds.

- **"polygon_table"**: a CSV file (or a data frame) of values keyed by a 2010 Census tract, block group, ZCTA or county. temporal: "static" (one row per place), "yearly" (a year column) or "daily" (a date column). key names the geography: "tract", "block_group", "zcta" or "county" when the key column has its usual name (such as GEOID10 or GEOID), else a list such as `list(boundary = "tract", column = "FIPS", year_column = "yr")` (`year_column` and `date_column` default to year and date).
- **"raster"**: GeoTIFF files on one grid. temporal: "static" (one file), "yearly", "biweekly" or "daily" (one file per period). The period of each file comes from its name (a year, or a date such as 20160101 or 2016-01-01) or from periods.
- **"features"**: point, line or polygon geometries (one geometry family per dataset) in a GeoPackage, a GeoJSON file or a zipped Shapefile, with a coordinate reference system. The variable is the distance in meters from each location to the nearest feature, named `<name>_dist_m` unless you name it. temporal: "static" or "yearly" (one file per year, or a year column).

Where. The registry is the directory root, else the option `spacescans.datasets_dir`, else the environment variable `SPACESCANS_DATASETS_DIR`, else the Python package's default, `platformdirs.user_data_dir("spacescans")` (on macOS `~/Library/Application Support/spacescans/datasets`). Names are unique within a registry. `ss_link()` and `ss_coverage()` look registered datasets up by the same rule, so set the option when you keep them elsewhere.

Visibility. "private" (default) or "deployment". The packages only record it; a web app deployment enforces it (a "deployment" dataset is offered to every user of that deployment).

Value

`ss_register_dataset()`, `ss_dataset()` and `ss_update_dataset()`: the dataset's record, a list of class `ss_dataset` (`id`, `name`, `kind`, `temporal`, `key`, `variables` (a data frame: `name`, `unit`, `description`), `source`, `version`, `description`, `license_note`, `visibility`, `owner`, `created_utc`, `files`, `checksum`, `spacescans_version`, `details`, `path` (its directory) and `variable_key`), with the registry directory as attribute root. `ss_datasets()`: a data frame with one row per dataset (variables as their names), attribute root. `ss_remove_dataset()`: the removed record, invisibly.

Examples

```
## Not run:
svi <- ss_register_dataset("~/data/svi_2020_tract.csv", name = "SVI 2020",
                           kind = "polygon_table", temporal = "static",
                           key = list(boundary = "tract", column = "FIPS"),
                           variables = "RPL_THEMES",
                           source = "CDC/ATSDR SVI 2020", version = "2020")

ss_datasets()
res <- ss_link(cohort, "ndi", custom = "SVI 2020")
ss_update_dataset("SVI 2020", visibility = "deployment")
ss_remove_dataset("SVI 2020")

## End(Not run)
```

ss_doctor

Check the spacescans setup

Description

Runs local checks and one bridge handshake, and reports each as ok, warn or fail: which interpreter is used and why, versions and protocol, Python extras, GDAL/GEOS/PROJ, macOS version and architecture, memory, machine-wide PROJ_*/GDAL_DATA variables (they are not passed to Python), whether the data, cache, environment or temporary directories sit in a cloud-synced folder, free disk space, the run directories under `tools::R_user_dir("spacescans", "cache")/runs` (those whose owner process no longer exists are removed, since they hold cohort data), and (when a data directory is known) how many catalog variables have their data in place ([ss_data_status\(\)](#)).

Usage

```
ss_doctor(data_dir = NULL)
```

Arguments

`data_dir` Data directory to check. Default: [ss_data_dir\(\)](#) when it is set.

Value

Invisibly, a data frame of class `ss_doctor` with columns `check`, `status` and `message`; it is printed.

Examples

```
## Not run:
ss_doctor()

## End(Not run)
```

`ss_features`*The feature dictionary*

Description

Describes every exposure column `ss_link()` can produce: its name, a short and a detailed description.

Usage

```
ss_features(variables = NULL)
```

Arguments

`variables` Variable keys to restrict to, or NULL (default) for every column.

Value

A data frame with `feature_name`, `short_description`, `detailed_description` (and the variable, when the bridge reports it).

Examples

```
## Not run:
ss_features("ndi")

## End(Not run)
```

`ss_fetch`*Download the data variables need*

Description

Downloads (resumably) what `ss_data_status()` reports missing. Shared files, such as the block-group boundaries several variables use, are downloaded once. Ctrl-C stops the download and keeps the partial files, so running the same call again resumes. On macOS the computer is kept awake while it runs (caffeinate).

Usage

```
ss_fetch(
  variables = NULL,
  datasets = NULL,
  data_dir = ss_data_dir(),
  accept_terms = character(),
  keep_archives = FALSE,
```

```

    dry_run = FALSE,
    progress = interactive()
  )

```

Arguments

variables	Variable keys; NULL with datasets = NULL fetches everything the catalog needs.
datasets	Dataset ids, instead of or in addition to variables.
data_dir	The data directory (default <code>ss_data_dir()</code>).
accept_terms	Ids of provider terms you accept for this download.
keep_archives	Keep the downloaded archives after extraction.
dry_run	Only report what would be downloaded (data-plan).
progress	Show progress (default: in interactive sessions).

Details

Some provider datasets require you to accept their terms. In an interactive session you are asked for each one; otherwise pass their ids in `accept_terms` (read them first with `ss_data_terms()`). Terms are accepted for this call only.

Value

Invisibly, the bridge's fetch report (with `dry_run = TRUE`, the plan, printed).

Examples

```

## Not run:
ss_fetch(c("ndi", "walkability"), dry_run = TRUE)
ss_fetch(c("ndi", "walkability"))

## End(Not run)

```

ss_install

Install the Python side of spacescans

Description

Builds (or updates) a private virtual environment with uv and installs the spacescans Python wheel that matches this R package, then checks it with the bridge. Nothing is downloaded or installed outside `envdir` and uv's cache, and the Python on PATH is never touched.

Usage

```
ss_install(
  extras = c("geo", "rda", "nc"),
  python = "3.12",
  envdir = NULL,
  wheel = NULL,
  wheelhouse = NULL,
  offline = FALSE,
  uv = NULL,
  force = FALSE,
  quiet = FALSE
)
```

Arguments

extras	Python extras to add: any of "geo", "rda", "nc", "hdf4" (TEMIS needs "hdf4"). Extras already installed stay installed.
python	Python version ("3.12") or path to an interpreter.
envdir	Environment directory. Default: option spacescans.envdir, then the environment variable SPACESCANS_ENVDIR, then tools::R_user_dir("spacescans", "data")/r-spacescans.
wheel	Path to a spacescans-*.whl (the Python package) to install.
wheelhouse	Directory of wheels to install from (offline sites).
offline	If TRUE, never use the network (needs wheelhouse or uv's cache).
uv	Path to the uv executable.
force	If TRUE, delete and rebuild the environment.
quiet	If TRUE, do not echo uv's output.

Details

Steps:

1. find uv: the uv argument, the SPACESCANS_UV environment variable, PATH, ~/.local/bin/uv, /opt/homebrew/bin/uv;
2. create the environment with `uv venv --python <python>` unless it exists (uv downloads a managed CPython if needed; set `UV_PYTHON_DOWNLOADS=never` and pass `python =` an interpreter path on locked-down machines);
3. install the dependencies from the bundled hash-pinned lock file when there is one for this platform (inst/python/locks/), then the wheel with `--no-deps`; without a lock, install `wheel[extras]` and let uv resolve. Only binary wheels are accepted for the dependencies, so nothing is compiled;
4. run the bridge once and write `env.json` in the environment.

The wheel is looked for in this order: the wheel argument, the wheel in the package's python/ directory (checked against `pin.json`), the wheelhouse directory, and the URL recorded in `pin.json` (downloaded and checked against its SHA-256). When there is none, as in an R package installed

from GitHub with `remotes::install_github()` or `pak`, or from `r-universe`, the `spacescans` Python package is installed from the source archive of the same GitHub commit, which those tools record in the installed package's `RemoteSha` field (`r-universe` records the repository as `RemoteUrl`; without a recorded commit, the tag of this package's version, `v<version>`, is used). The pipeline is pure Python, so building it compiles nothing.

Value

Invisibly, the bridge's capabilities for the new environment.

Examples

```
## Not run:
ss_install()
ss_install(extras = "hdf4")  # add TEMIS support

## End(Not run)
```

ss_job

Background jobs

Description

`ss_link(wait = FALSE)` returns a job handle at once; these functions follow it. A job's progress events, log, request and result live in its run directory.

Usage

```
ss_job_status(job)

ss_job_wait(job, timeout = Inf, progress = interactive())

ss_job_cancel(job, grace = getOption("spacescans.interrupt_grace", 10))

ss_job_result(job, wait = TRUE, progress = interactive())

ss_job_attach(run_dir, cohort = NULL, cols = NULL)
```

Arguments

<code>job</code>	A job from <code>ss_link(wait = FALSE)</code> or <code>ss_job_attach()</code> .
<code>timeout</code>	Seconds to wait; <code>Inf</code> (default) waits until the job ends.
<code>progress</code>	Show a progress line.
<code>grace</code>	Seconds to wait after <code>SIGINT</code> before killing the process tree (default: <code>getOption("spacescans.interrupt_grace", 10)</code>).
<code>wait</code>	For <code>ss_job_result()</code> : wait for the job to finish first.

run_dir	The job's run directory.
cohort	For <code>ss_job_attach()</code> of a link job: the cohort data frame the job was started with. It is compared with the cohort file in the run directory; an episode-level result is joined back onto it, and a participant-level result gets <code>pid</code> in the type of its <code>id</code> column. Without it, <code>ss_job_result()</code> returns, at episode level, the exposures with an <code>episode_id</code> column (the 0-based row number in the cohort) and, at participant level, the participant table with <code>pid</code> as text.
cols	Column mapping used when the job was started (see <code>ss_link()</code>).

Details

- `ss_job_status()` reports the state ("running", "succeeded", "failed", "interrupted" or "crashed"), the current step and the progress.
- `ss_job_wait()` waits for the job, showing progress. Pressing Ctrl-C stops waiting but leaves the job running (use `ss_job_cancel()` to stop it).
- `ss_job_cancel()` asks the job to stop (SIGINT), waits up to `grace` seconds, then kills its whole process tree. Completed C3 steps stay in the cache, so running the same link again is fast.
- `ss_job_result()` waits (unless `wait = FALSE`) and returns what the finished command returns (for `ss_link()`, the participant table or the linked cohort, by `output_level`), or raises its error.
- `ss_job_attach()` finds a job again from its run directory, for example after R restarted. Only jobs started with an explicit `run_dir` survive the R session: a run directory the package chose itself is tied to the session that started the job (the bridge stops with it, and the directory is removed by the next sweep once that session is gone).

Value

`ss_job_status()`: a list of class `ss_job_status`. `ss_job_wait()` and `ss_job_cancel()`: the status, invisibly. `ss_job_result()`: the command's result. `ss_job_attach()`: a job.

Examples

```
## Not run:
job <- ss_link(cohort, c("ndi", "walkability"), wait = FALSE,
              run_dir = "~/ss-runs/ndi-walk")
ss_job_status(job)
res <- ss_job_result(job)

# after restarting R
job <- ss_job_attach("~/ss-runs/ndi-walk", cohort = cohort)
res <- ss_job_result(job)

## End(Not run)
```

ss_link

*Link a cohort to catalog variables***Description**

Computes exposures for every location episode of a cohort and returns them per participant (the default) or per location episode.

Usage

```
ss_link(
  cohort,
  variables,
  output_level = c("participant", "episode"),
  data_dir = ss_data_dir(),
  buffer_m = 270L,
  raster_res_m = 25L,
  custom = NULL,
  cols = NULL,
  cache = TRUE,
  threads = NULL,
  memory_limit = NULL,
  on_error = c("stop", "continue"),
  progress = interactive(),
  keep = FALSE,
  run_dir = NULL,
  wait = TRUE,
  workers = NULL
)
```

Arguments

cohort	A data frame with one row per location episode, or the path of a CSV or Parquet file.
variables	Catalog variable keys, for example <code>c("ndi", "walkability")</code> (see ss_catalog()). The variables of custom datasets are added automatically, so <code>character()</code> links the custom datasets alone.
output_level	"participant" (default): one row per participant, each variable averaged over the participant's location episodes weighted by contributing days (see Details). "episode": one row per location episode, your data frame with the exposure columns added.
data_dir	The data directory (default ss_data_dir()).
buffer_m	Buffer radius around each location in meters (whole number).
raster_res_m	Raster resolution in meters used by areal C3 steps.

custom	Your own datasets: a dataset registered with <code>ss_register_dataset()</code> (its name, id or variable key, or the record), a definition from <code>ss_custom_polygon()</code> or <code>ss_custom_raster()</code> , or a list (or character vector) of them, linked in that order.
cols	Your column names for the required columns, for example <code>c(pid = "participant_id", startDate = "start", endDate = "end", longitude = "lon", latitude = "lat")</code> .
cache	TRUE (default) keeps C3 weights in the user cache directory (<code>tools::R_user_dir("spacescans", "cache")</code>), see <code>ss_cache_info()</code> , so the same cohort and buffer are not recomputed; FALSE disables it; a path uses that directory as the cache root.
threads, memory_limit	DuckDB worker threads and memory cap (for example "8GB") for the pipeline steps.
on_error	"stop" stops at the first failing data source; "continue" runs the others and returns what succeeded (with a warning naming the failures).
progress	Show progress (default: in interactive sessions).
keep	Keep the run directory after success.
run_dir	Run directory to use (created with mode 0700; must be empty). Default: a fresh directory under <code>tools::R_user_dir("spacescans", "cache")/runs</code> , owned by this session.
wait	TRUE (default) waits for the result; FALSE returns a job at once (see <code>ss_job_status()</code>). Only a job with an explicit <code>run_dir</code> survives the R session.
workers	Worker processes per pipeline step for the work DuckDB does not do: the spatial computation of every catalog source (polygon overlap, raster weights, nearest-feature distances) and the decoding of the ACAG NetCDF and TEMIS HDF4 files. Default NULL, one process. The result is the same with any number; each worker adds memory.

Details

A cohort is time-referenced geolocation data: one row per location episode, that is a participant, a location (longitude and latitude) and the first and last day spent there. A participant who moved has one row per location.

Output level. With `output_level = "participant"` (the default) the result has one row per participant: for each variable, the mean of the participant's episode values weighted by the days that contribute to them. For participant i and variable v it is

$$\bar{x}_{iv} = \frac{\sum_{e \in E_i} \sum_{t \in T_{ev}} d_{et} z_{etv}}{\sum_{e \in E_i} \sum_{t \in T_{ev}} d_{et}}$$

where E_i are the participant's episodes, z_{etv} is the spatial estimate (buffer-weighted mean, or distance to the nearest feature) for episode e in source period t , T_{ev} are the periods that overlap the episode and have an estimate, and d_{et} is the number of days of the overlap (first and last day included). A time-invariant source counts every day of the episode. The contributing days are counted per variable, and a participant with no contributing day gets a missing value. Where two location episodes of a participant overlap in time, each contributes its own days: a calendar day both cover counts once for each location (trim the overlaps first if a day should count once).

Variables derived after aggregation come from the participant's means: the FARA flags (such as LILATracts_1And10) are recoded from the participant's mean rates and urban share, so they stay 0 or 1, and an ACAG_nbm (non-biomass) column is the participant's mean of the species minus the participant's mean of its biomass part (_bm). With `output_level = "episode"` the result is the cohort itself with the exposure columns added, one row per location episode, as in spacescans 0.2.

What happens, in order:

1. The Python bridge is checked once per session (versions, protocol).
2. The cohort is checked in R: `cols` maps your column names; PATID in any case stands in for pid; pid becomes text; dates become Date (date-times are converted with a warning; text dates must use one format, 2017-08-19 or 8/19/2017); every row needs a pid (a blank one is an error, so no row is left out of a participant table) and coordinates that are present and in range. Problems are reported with row numbers, never values.
3. Only the five columns pid, startDate, endDate, longitude and latitude are written, to `cohort.parquet` in a private run directory (mode 0700, under `tools::R_user_dir("spacescans", "cache")/runs` unless you give `run_dir`). Your other columns never leave R.
4. The data directory is checked (`ss_data_status()`): when data is missing the error says which `ss_fetch()` call to run, and when a Python extra is missing, which `ss_install()` call. `ss_link()` never downloads anything.
5. The Python side runs the pipeline steps, showing progress. Ctrl-C asks it to stop, waits up to 10 seconds (option `spacescans.interrupt_grace`) and then kills its process tree. Completed C3 steps stay in the cache.
6. Participant level: the table has pid (with the type of your id column) and one column per exposure, one row per participant in the order they first appear in the cohort, including participants without any value. Episode level: the exposures are joined back onto your data frame by row (the `episode_id` of row i is $i - 1$); its column types, class and attributes are kept, and an exposure column whose name is already taken gets the suffix `_<experiment>_dup`, as in the web app.
7. With `keep = FALSE` the run directory is deleted after success; cache entries stay. After an error it is kept and its path printed. A run directory the package chose itself is recorded as owned by this R session; should the session die, the next package load, `ss_doctor()` or `ss_cache_clear()` (in any session of yours) deletes it.

Value

Participant level: a data frame with pid and one column per exposure, one row per participant (a `data.table` when `cohort` is one). Episode level: the cohort data frame with the exposure columns added. Both carry the attributes `output_level`, `features` (the feature dictionary of the exposure columns), `coverage` (the linkage rates: per variable, the rows of the result, the rows with a value and their percentage; participants or episodes, by level), `provenance` (versions, code fingerprint, data receipts, cache keys, threads, workers, platform; no cohort values) and `spacescans` (steps, failed and skipped sources, warnings). With `wait = FALSE`, a job.

See Also

`ss_coverage()` estimates, per location episode, how much of the cohort a variable covers before you link.

Examples

```
## Not run:
ss_data_dir("~/ss-data")
res <- ss_link(cohort, c("ndi", "walkability"))      # one row per participant
attr(res, "coverage")                               # linkage rates
attr(res, "features")
ep <- ss_link(cohort, "ndi", output_level = "episode") # one row per location episode

## End(Not run)
```

ss_quickstart	<i>Run the bundled quickstart demo</i>
---------------	--

Description

Runs a small C3 weight computation (`boundary_overlap_fast`) on the synthetic sample counties and locations that come with the Python package. Needs the `geo extra`. Useful to check that an installation works; it runs one spatial-weighting step, not a linkage.

Usage

```
ss_quickstart(
  output_dir = NULL,
  threads = NULL,
  read = TRUE,
  progress = interactive(),
  verbose = FALSE
)
```

Arguments

<code>output_dir</code>	Root for a relative output path. Default: a fresh directory under <code>tempdir()</code> , removed when the R session ends.
<code>threads</code>	DuckDB worker threads.
<code>read</code>	If TRUE (default), return the output table; otherwise return the run summary.
<code>progress</code>	Show a progress line (default: in interactive sessions).
<code>verbose</code>	Print the pipeline's own log lines.

Value

As `ss_run()`.

Examples

```
## Not run:
w <- ss_quickstart()
head(w)

## End(Not run)
```

ss_read_config

Read and write pipeline configs as YAML

Description

ss_read_config() parses a YAML config into a list; ss_write_config() writes a list as YAML the way spacescans-web does (yaml.safe_dump(sort_keys = FALSE)), wrapping length-one vectors into lists where the schema needs a list. Both use the Python bridge, so no R YAML package is needed.

Usage

```
ss_read_config(path)

ss_write_config(config, path)
```

Arguments

path	Path of the YAML file.
config	The config as a named list.

Value

ss_read_config(): the config as a list. ss_write_config(): invisibly, the YAML text.

Examples

```
## Not run:
cfg <- ss_read_config("c3_bg.yaml")
cfg$buffer$buffer_m <- 500L
ss_write_config(cfg, "c3_bg_500m.yaml")

## End(Not run)
```

ss_read_result	<i>Read a pipeline output table</i>
----------------	-------------------------------------

Description

Reads a Parquet output with **nanoparquet** (no Arrow needed). Episode-level C4 outputs carry the episode id in a column named geoid; it is renamed to episode_id so it can be joined back to the cohort rows (episode_id is the 0-based input row number).

Usage

```
ss_read_result(path, episode = NULL)
```

Arguments

path	Path to a .parquet (or .csv) file.
episode	TRUE to rename geoid to episode_id, FALSE to leave it, NULL (default) to rename only when the table has both PATID and geoid columns (an episode-level C4 output).

Details

64-bit integer columns are read as R doubles by nanoparquet, which is exact up to 2^{53} ; geoid/episode_id columns whose values fit are converted to integer.

Value

A data frame.

Examples

```
## Not run:  
df <- ss_read_result("output/c4_ndi.parquet")  
  
## End(Not run)
```

ss_run	<i>Run one pipeline config</i>
--------	--------------------------------

Description

Runs a single C3 or C4 config (the tier-1 interface) in a separate Python process. Progress is shown while it runs; Ctrl-C stops the Python process tree. On success the output is read back into R.

Usage

```
ss_run(
  config,
  data_dir = NULL,
  output_dir = NULL,
  threads = NULL,
  memory_limit = NULL,
  read = TRUE,
  progress = interactive(),
  verbose = FALSE,
  strict = FALSE,
  workers = NULL
)
```

Arguments

config	Path to a YAML config, or the config as a named list (see ss_templates() and ss_config_schema()).
data_dir	Root for relative input paths. Default: the SPACESCANS_DATA_DIR environment variable, then the config's base_dir.
output_dir	Root for a relative output.path. Default: a fresh directory under tempdir() , removed when the R session ends.
threads, memory_limit	DuckDB worker threads and memory cap (for example "8GB"). They cap DuckDB only: C3 patterns and the pandas-based FAQSD and TEMIS readers do not run inside DuckDB.
read	If TRUE (default), return the output table; otherwise return the run summary.
progress	Show a progress line (default: in interactive sessions).
verbose	Print the pipeline's own log lines.
strict	If TRUE, unknown config keys are an error.
workers	Worker processes for the work DuckDB does not do, where the config's pattern supports it (boundary_overlap_fast, grid_weights, nhd_proximity, tiger_proximity, acag_multi, and the HDF4 path of the TEMIS reader); written to the config's engine.workers. Default NULL, one process.

Value

With read = TRUE, a data frame (see [ss_read_result\(\)](#)) with attribute spacescans_run holding the run summary. Otherwise the summary: a list with output, rows, columns, seconds, maxrss_bytes, engine, the resolved config and more.

Examples

```
## Not run:
cfg <- ss_templates("c4/zbp")
res <- ss_run(cfg, data_dir = "~/ss-data", threads = 4)

## End(Not run)
```

ss_templates

Config templates shipped with the Python package

Description

Two sets exist: the 25 catalog templates that `ss_link()` renders for the catalog variables (kind = "catalog", the default; listed with their SHA-256), and the 5 starter configs for writing your own C3/C4 configs for `ss_run()` (kind = "init").

Usage

```
ss_templates(name = NULL, kind = c("catalog", "init"))
```

Arguments

name	NULL (default) to list the templates, or a template name such as "c3/county" to get that template.
kind	"catalog" or "init".

Value

Without name, a data frame with one row per template (name, kind, file, and for catalog templates sha256, ...). With name, the template as a list (attribute text holds the YAML text).

Examples

```
## Not run:
ss_templates()
ss_templates(kind = "init")
cfg <- ss_templates("c4/zbp", kind = "init")

## End(Not run)
```

ss_use_python

Choose the Python interpreter spacescans uses

Description

The interpreter is chosen in this order, and the first one found is used:

1. the one set by `ss_use_python()` in this session;
2. the environment variable SPACESCANS_PYTHON (a python executable or a virtual environment directory; IT can set it in `Renviron.site`);
3. the environment built by `ss_install()`.

Usage

```
ss_use_python(python, check = TRUE)
```

```
ss_python()
```

Arguments

python	Path to a python executable, or to a virtual environment directory (its bin/python is used).
check	If TRUE (default), start the bridge once to make sure the interpreter has a compatible spacescans package, and fail otherwise.

Details

The python on PATH, conda environments and reticulate's choice are never used implicitly.

Value

ss_use_python() invisibly returns the bridge's capabilities (or NULL when check = FALSE). ss_python() returns a list with path and source (why it was chosen).

Examples

```
## Not run:
ss_use_python("~/venvs/spacescans/bin/python")
ss_python()

## End(Not run)
```

ss_validate

Validate a pipeline config without running it

Description

Resolves every path (relative paths against data_dir and output_dir) and validates the config against the schema. Length-one R vectors are accepted where the schema wants a list (value_cols = "pm25"). Keys the schema does not know are reported, because the pipeline would ignore them.

Usage

```
ss_validate(config, data_dir = NULL, output_dir = NULL, strict = FALSE)
```

Arguments

config	Path to a YAML config, or the config as a named list.
data_dir	Root for relative input paths. Default: the SPACESCANS_DATA_DIR environment variable, then the config's base_dir.
output_dir	Root for a relative output.path.
strict	If TRUE, unknown keys are an error instead of a warning.

Value

The validated config as a list, with every default filled in and paths absolute; attribute `unknown_keys` lists ignored keys.

Examples

```
## Not run:
cfg <- ss_validate("c4/zbp.yaml", data_dir = "~/ss-data")
cfg$exposure$value_cols

## End(Not run)
```

ss_version	<i>Versions of the R package, the Python package and the bridge protocol</i>
------------	--

Description

Versions of the R package, the Python package and the bridge protocol

Usage

```
ss_version()
```

Value

A list of class `ss_version` with `r_package`, `python_package`, `protocol`, `code_fp` (fingerprint of the Python code that runs), `python` (interpreter version), `executable` and `source` (why that interpreter was chosen, see [ss_use_python\(\)](#)).

Examples

```
## Not run:
ss_version()

## End(Not run)
```

Index

difftime, [4](#)

spacescans-conditions, [3](#)

ss_cache_clear(ss_cache_info), [4](#)

ss_cache_clear(), [22](#)

ss_cache_info, [4](#)

ss_cache_info(), [21](#)

ss_catalog, [5](#)

ss_catalog(), [6, 20](#)

ss_config_schema, [5](#)

ss_config_schema(), [26](#)

ss_coverage, [6](#)

ss_coverage(), [13, 22](#)

ss_custom_polygon, [7](#)

ss_custom_polygon(), [6, 21](#)

ss_custom_raster(ss_custom_polygon), [7](#)

ss_custom_raster(), [6, 21](#)

ss_data_dir, [8](#)

ss_data_dir(), [9, 10, 14, 16, 20](#)

ss_data_import(ss_data_verify), [10](#)

ss_data_status, [9](#)

ss_data_status(), [5, 10, 14, 15, 22](#)

ss_data_terms(ss_data_verify), [10](#)

ss_data_terms(), [16](#)

ss_data_verify, [10](#)

ss_data_verify(), [3](#)

ss_dataset(ss_datasets), [11](#)

ss_datasets, [11](#)

ss_doctor, [14](#)

ss_doctor(), [4, 22](#)

ss_features, [15](#)

ss_fetch, [15](#)

ss_fetch(), [3, 8, 22](#)

ss_install, [16](#)

ss_install(), [3, 22, 27](#)

ss_job, [18](#)

ss_job_attach(ss_job), [18](#)

ss_job_cancel(ss_job), [18](#)

ss_job_result(ss_job), [18](#)

ss_job_status(ss_job), [18](#)

ss_job_status(), [21](#)

ss_job_wait(ss_job), [18](#)

ss_link, [20](#)

ss_link(), [4, 6–8, 11, 13, 15, 19, 27](#)

ss_python(ss_use_python), [27](#)

ss_quickstart, [23](#)

ss_read_config, [24](#)

ss_read_result, [25](#)

ss_read_result(), [26](#)

ss_register_dataset(ss_datasets), [11](#)

ss_register_dataset(), [6, 8, 21](#)

ss_remove_dataset(ss_datasets), [11](#)

ss_run, [25](#)

ss_run(), [23, 27](#)

ss_templates, [27](#)

ss_templates(), [26](#)

ss_update_dataset(ss_datasets), [11](#)

ss_use_python, [27](#)

ss_use_python(), [29](#)

ss_validate, [28](#)

ss_version, [29](#)

ss_write_config(ss_read_config), [24](#)

tempdir(), [7, 23, 26](#)