Package: pdokr (via r-universe)

October 18, 2026

Title Access Open Geodata from the Dutch 'PDOK' Platform

Version 0.1.0

Description Tools to discover, download, and spatially filter open geographic data from 'PDOK' (Publieke Dienstverlening Op de Kaart), the national geodata platform of the Netherlands. Datasets and their layers are searched and loaded as vector simple feature ('sf') objects through 'OGC' API Features endpoints, with automatic pagination and explicit coordinate reference system handling. Loaded layers can be filtered by any polygon area, and addresses or place names can be geocoded through the 'PDOK' location server. The focus is on vector feature data; raster, tile, and coverage services are out of scope. See <https://www.pdok.nl/> for more information about the platform and its services.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 8.0.0

Imports cli, httr2, rlang, sf, tibble

Suggests httptest2, knitr, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

URL https://github.com/coeneisma/pdokr, https://coeneisma.github.io/pdokr/

BugReports https://github.com/coeneisma/pdokr/issues

Language en-US

Config/Needs/website rmarkdown, tmap, dplyr, mapgl, leaflet

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

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

Date/Publication 2026-07-22 11:52:15 UTC

RemoteUrl https://github.com/coeneisma/pdokr

RemoteRef HEAD

RemoteSha 3e009df8941259ac709ed4e1fa4789f76aca9196

Contents

pdok_basemap 2
pdok_filter_by 3
pdok_geocode 4
pdok_list_datasets 6
pdok_list_layers 6
pdok_read 7
pdok_reverse_geocode 9
pdok_search_datasets 10
pdok_search_layers 10

Index **12**

`pdok_basemap` *A PDOK basemap to use as a map background*

Description

Returns the URL of the official PDOK 'BRT Achtergrondkaart' (or aerial imagery) for use as the background of a map. Nothing is downloaded — the function only constructs the URL, so it works offline and instantly. Hand the result to any mapping package: `tmap` and `leaflet` take the raster tile URL, `maplibre`/`mapgl` take the vector style URL.

Usage

```
pdok_basemap(style = "standaard", format = c("raster", "vector"))
```

Arguments

`style`	The basemap style. For `format = "raster"`: one of `"standaard"`, `"grijs"`, `"pastel"`, `"water"`, or `"luchtfoto"` (aerial imagery). For `format = "vector"`: one of `"standaard"`, `"zonder_labels"`, `"luchtfoto"`, or `"darkmode"`.
`format`	`"raster"` (the default) returns a WMTS tile-URL template (`{z}/{x}/{y}`, in Web Mercator / EPSG:3857) that works with `tmap`, `leaflet` and `maplibre`/`mapgl`. `"vector"` returns a Mapbox GL style URL for `maplibre`/`mapgl`.

Value

A single string: a raster tile-URL template, or a vector style URL.

Attribution

The map data is © Kadaster / PDOK. Show this attribution on any map that uses the basemap.

Examples

```
# Raster tile URL (tmap / leaflet)
pdok_basemap()
pdok_basemap("grijs")

# Vector style URL (maplibre / mapgl)
pdok_basemap("standaard", format = "vector")
```

---

pdok_filter_by *Spatially filter an sf layer by any polygon*

Description

Keeps the features of `data` that relate to `filter_geometry` under a spatial predicate. It does what `sf::st_filter()` does, and additionally reconciles coordinate reference systems for you: `filter_geometry` is transformed to the CRS of `data` before filtering.

Usage

```
pdok_filter_by(data, filter_geometry, predicate = "intersects")
```

Arguments

`data`	An sf object to filter (for example a layer loaded with `pdok_read()`).
`filter_geometry`	An `sf` or `sfc` object whose geometry defines the area of interest.
`predicate`	The spatial relationship to test, one of `"intersects"`, `"within"`, `"contains"`, `"overlaps"`, `"touches"`, `"crosses"`, `"covers"`, `"covered_by"`, or `"disjoint"`. See the *Spatial predicates* section below, and sf::geos_binary_pred for the full definitions.

Details

`filter_geometry` can be *any* polygon: a municipality from `pdok_read()` on the CBS administrative boundaries, a nature reserve, a water-authority area, a hand-drawn polygon, or another PDOK layer.

The plain-`sf` equivalent is `data[filter_geometry, , op = sf::st_intersects]` (after matching CRS); use that if you prefer to drop down to `sf`.

Value

An sf object: the subset of `data` whose features satisfy `predicate` with respect to `filter_geometry`. A zero-row `sf` is returned (silently) when no feature matches.

Spatial predicates

The `predicate` selects *how* a feature must relate to `filter_geometry` to be kept. Each value calls the matching `sf` predicate function `sf::st_<predicate>()` (so `"within"` uses `sf::st_within()`, `"intersects"` uses `sf::st_intersects()`, and so on). The common choices:

- `"intersects"` (default) — the feature touches `filter_geometry` in any way (overlaps, is inside, or shares a boundary). The most permissive; the usual choice for "everything in this area".
- `"within"` — the feature lies *entirely inside* `filter_geometry`. Use this to exclude features that only stick partly into the area.
- `"contains"` — the feature entirely *encloses* `filter_geometry` (the inverse of `"within"`).
- `"disjoint"` — the feature does *not* touch `filter_geometry` at all (everything outside the area).

`"overlaps"`, `"touches"`, `"crosses"`, `"covers"` and `"covered_by"` are the remaining, more specialized DE-9IM relationships. For the precise definition of each, see the `sf` help page sf::geos_binary_pred (the topic that documents `st_intersects()`, `st_within()`, and the rest).

See Also

sf::geos_binary_pred for the definition of each spatial predicate; `sf::st_filter()`, the `sf` filter this wraps; and `pdok_read()`, whose `filter_by` argument applies this filter while loading.

Examples

```
# All national parks that intersect the province of Utrecht
utrecht <- pdok_read(
  "cbs/gebiedsindelingen", "provincie_gegeneraliseerd",
  datetime = 2024
)
utrecht <- utrecht[utrecht$statnaam == "Utrecht", ]
parks <- pdok_read("rvo/nationale-parken-geharmoniseerd", "protectedsite")
pdok_filter_by(parks, utrecht)
```

---

pdok_geocode *Geocode addresses or place names with the PDOK Locatieserver*

Description

Looks up addresses, place names, postcodes, municipalities, provinces and more through the 'PDOK' Locatieserver, returning the results as a simple feature collection. Point geometry is returned for addresses and places, and boundary polygons for administrative areas such as municipalities — so a result drops straight into the `filter_by` argument of `pdok_read()`.

Usage

```
pdok_geocode(query, type = NULL, crs = NULL, limit = 1)
```

Arguments

query	A character vector of one or more non-empty search strings, e.g. "Park Arenberg 88, De Bilt".
type	Optional result type to restrict to, one of "adres", "postcode", "weg", "woonplaats", "gemeente", "provincie", "buurt", "wijk", "perceel", "hectometerpaal", or "appartementsrecht". NULL (the default) returns the best matches of any type, ranked by the service's relevance score. Use type to disambiguate names that exist in several categories (for example "Utrecht" is both a municipality and a province).
crs	Optional output CRS as an EPSG code (e.g. 28992). NULL keeps the source CRS (CRS84, lon/lat).
limit	Maximum number of results to return per query (default 1).

Details

query may be a vector, geocoding many locations in one call (for example a column of addresses). Each result row carries a query column with the input it came from, so the output maps back to the input even when a query returns several candidates or none.

Value

An sf object with one row per match and a query column identifying the input each row came from. All non-geometry fields the service returns are kept as columns (with query, weergavenaam, type, score, and the administrative names first); the geometry is a point for addresses and places and a polygon for administrative areas. Queries that match nothing are dropped (with a warning), so the result can have fewer rows than query has elements; a zero-row sf is returned when nothing matches at all.

See Also

pdok_reverse_geocode() for the reverse lookup (coordinates to the nearest address), and pdok_read(), whose filter_by argument accepts the result.

Examples

```
# An address: a point
pdok_geocode("Park Arenberg 88, De Bilt")

# A municipality: a boundary polygon
pdok_geocode("De Bilt", type = "gemeente")

# Several addresses in one call; the `query` column maps rows to inputs
pdok_geocode(c("Domplein 1, Utrecht", "Coolsingel 40, Rotterdam"))
```

pdok_list_datasets *List PDOK datasets*

Description

Returns the full table of contents of datasets offered through the 'PDOK' OGC API Features platform, fetched live from https://api.pdok.nl/index.json.

Usage

```
pdok_list_datasets()
```

Value

A tibble with one row per dataset and the columns `id` (the identifier passed to `pdok_list_layers()` and `pdok_read()`), `name`, `description`, `keywords` (a list-column of character vectors), `services`, `owner`, and `ogc_url`.

See Also

`pdok_search_datasets()` to filter this list.

Examples

```
pdok_list_datasets()
```

pdok_list_layers *List the layers within a PDOK dataset*

Description

Lists the layers (OGC API Features collections) offered by a dataset.

Usage

```
pdok_list_layers(dataset)
```

Arguments

`dataset`	A dataset id from `pdok_list_datasets()` (e.g. `"cbs/gebiedsindelingen"`), or a raw OGC API base URL.

Value

A tibble with one row per layer and the columns `dataset` (the dataset id, echoing the input so each row works directly with `pdok_read()`), `layer` (the layer identifier), `title`, `description`, `start_date` and `end_date` (the temporal extent the layer covers, as `Dates`; `end_date` is `NA` when the layer is ongoing), `crs` (a list-column of available EPSG codes), `storage_crs` (the EPSG code the data is stored in), and `bbox` (a list-column of named numeric extents `c(xmin, ymin, xmax, ymax)` in CRS84).

See Also

`pdok_search_layers()` to filter this list, `pdok_list_datasets()` for the datasets.

Examples

```
pdok_list_layers("cbs/gebiedsindelingen")
```

---

pdok_read *Read a PDOK layer as an sf object*

Description

Loads a layer from PDOK as a simple feature collection over the OGC API Features service, handling pagination automatically.

Usage

```
pdok_read(
  dataset,
  layer,
  bbox = NULL,
  filter_by = NULL,
  predicate = "intersects",
  datetime = NULL,
  crs = NULL,
  max_features = NULL
)
```

Arguments

`dataset`	A dataset id from `pdok_list_datasets()` (e.g. `"cbs/gebiedsindelingen"`), or a raw OGC API base URL.
`layer`	A layer id from `pdok_list_layers()`.
`bbox`	Optional server-side bounding-box pre-filter: a numeric vector `c(xmin, ymin, xmax, ymax)` (assumed CRS84) or an `sf`/`sfc`/`bbox` object whose extent is used.

`filter_by`	Optional `sf`/`sfc` geometry to filter the result by. Its bounding box is used as a cheap server-side pre-filter, and the result is then filtered exactly with `pdok_filter_by()`. This is the one-call form of the load-then-filter workflow. It is usually a polygon (e.g. a municipality), but a point works too: filtering an area layer by a point returns the feature that contains it (for example the municipality an address falls in).
`predicate`	Spatial predicate for `filter_by`, passed to `pdok_filter_by()` (default `"intersects"`). See its *Spatial predicates* section for the available options, and sf::geos_binary_pred for their definitions.
`datetime`	Optional temporal filter: a single year (e.g. `2026`, mapped to a mid-year instant), an OGC datetime string, or an interval such as `"2020-01-01/2025-12-31"`.
`crs`	Optional output CRS as an EPSG code (e.g. `28992` for RD New). `NULL` keeps the source CRS.
`max_features`	Optional cap on the number of features returned.

Details

By default the data is returned in the coordinate reference system the service provides (lon/lat, CRS84, for the OGC path). Set `crs` to receive the data in another CRS; the transformation is done client-side with `sf::st_transform()`.

Value

An sf object with one row per feature, the layer's attribute columns, and a geometry column. A zero-row `sf` is returned (with a warning) when nothing matches.

See Also

`pdok_list_layers()` to find layer ids.

Examples

```
# A whole layer: the Dutch national parks
parks <- pdok_read("rvo/nationale-parken-geharmoniseerd", "protectedsite")

# Municipalities for 2024, in RD New (EPSG:28992)
pdok_read(
  "cbs/gebiedsindelingen", "gemeente_gegeneraliseerd",
  datetime = 2024, crs = 28992, max_features = 5
)

# One-call area filter: national parks within the province of Utrecht
provinces <- pdok_read(
  "cbs/gebiedsindelingen", "provincie_gegeneraliseerd", datetime = 2024
)
utrecht <- provinces[provinces$statnaam == "Utrecht", ]
parks_utrecht <- pdok_read(
  "rvo/nationale-parken-geharmoniseerd", "protectedsite",
  filter_by = utrecht
)
```

pdok_reverse_geocode *Reverse geocode coordinates to the nearest address with the PDOK Locatieserver*

Description

Finds the address, road or place nearest to each point, through the 'PDOK' Locatieserver reverse service — the inverse of pdok_geocode(). Give it sf points (in any CRS); it returns the nearest match(es) as an sf, including an afstand column with the distance in meters.

Usage

```
pdok_reverse_geocode(points, type = NULL, crs = NULL, limit = 1)
```

Arguments

points	An sf or sfc object of one or more points, in any CRS. The coordinates are transformed to lon/lat internally for the query.
type	Optional result type to restrict to (see pdok_geocode() for the list); NULL (the default) returns the nearest match of any type.
crs	Optional output CRS as an EPSG code (e.g. 28992). NULL keeps the source CRS (CRS84, lon/lat).
limit	Maximum number of results to return per point (default 1, the single nearest match).

Value

An sf object with one row per match and a point_id column giving the row index of the input point each match came from. The afstand column holds the distance in meters; all other non-geometry fields the service returns are kept too. Points that match nothing are dropped (with a warning); a zero-row sf is returned when nothing matches at all.

See Also

pdok_geocode() for the forward lookup (address to coordinates).

Examples

```
# A point in the center of Utrecht: the nearest address
pt <- sf::st_sfc(sf::st_point(c(5.121, 52.090)), crs = 4326)
pdok_reverse_geocode(pt)
```

pdok_search_datasets *Search PDOK datasets*

Description

Filters the dataset registry from pdok_list_datasets() by a case-insensitive partial match. The query is matched against each dataset's identifier, name, description, and keywords.

Usage

```
pdok_search_datasets(query)
```

Arguments

query	A single non-empty string to search for, e.g. "gemeente".

Value

A tibble with the same columns as pdok_list_datasets(), containing only the matching rows (zero rows when nothing matches).

See Also

pdok_list_datasets() for the full list.

Examples

```
pdok_search_datasets("gemeente")
```

pdok_search_layers *Search the layers within a PDOK dataset*

Description

Filters the layers from pdok_list_layers() by a case-insensitive partial match against each layer's identifier, title, and description.

Usage

```
pdok_search_layers(dataset, query)
```

Arguments

dataset	A dataset id from pdok_list_datasets() (e.g. "cbs/gebiedsindelingen"), or a raw OGC API base URL.
query	A single non-empty string to search for, e.g. "gemeente".

Value

A tibble with the same columns as `pdok_list_layers()`, containing only the matching rows (zero rows when nothing matches).

See Also

`pdok_list_layers()` for the full list.

Examples

```
pdok_search_layers("cbs/gebiedsindelingen", "gemeente")
```

Index

pdok_basemap, 2
pdok_filter_by, 3
pdok_filter_by(), *8*
pdok_geocode, 4
pdok_geocode(), *9*
pdok_list_datasets, 6
pdok_list_datasets(), *6, 7*, *10*
pdok_list_layers, 6
pdok_list_layers(), *6–8*, *10, 11*
pdok_read, 7
pdok_read(), *3–7*
pdok_reverse_geocode, 9
pdok_reverse_geocode(), *5*
pdok_search_datasets, 10
pdok_search_datasets(), *6*
pdok_search_layers, 10
pdok_search_layers(), *7*

sf, *3*, *5*, *8, 9*
sf::geos_binary_pred, *3, 4*, *8*
sf::st_filter(), *3, 4*
sf::st_intersects(), *4*
sf::st_transform(), *8*
sf::st_within(), *4*

tibble, *6, 7*, *10, 11*