Package: addr (via r-universe)

January 10, 2025

Title Clean, Parse, Harmonize, Match, and Geocode Messy Real-World Addresses

Version 0.6.0

Description Addresses that were not validated at the time of collection are often heterogenously formatted, making them difficult to compare or link to other sets of addresses. The addr package is designed to clean character strings of addresses, use the `usaddress` library to tag address components, and paste together select components to create a normalized address. Normalized addresses can be hashed to create hashdresses that can be used to merge with other sets of addresses.

URL https://github.com/cole-brokamp/addr, https://cole-brokamp.github.io/addr/

BugReports https://github.com/cole-brokamp/addr/issues

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), sf, s2, tidyr

Imports purrr, cli, stringr, dplyr, glue, fs, tibble, rlang, vctrs, methods, stringdist, zeallot

SystemRequirements Cargo (Rust's package manager), rustc

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first addr_match*, s2_join_tiger_bg

Depends R (>= 3.5.0)

Config/rextendr/version 0.3.1

Config/pak/sysreqs make libicu-dev

Repository https://geomarker-io.r-universe.dev

RemoteUrl https://github.com/geomarker-io/addr

RemoteRef HEAD

RemoteSha a814cfa2008f0563726bb1740af911d7f3023d8e

Contents

`addr` *Create a new addr vector*

Description

An addr vector is created by converting messy, real-world mailing addresses in a character vector into a list of standardized address tags that behaves like a vector. `addr()` (and `as_addr()`) vectors are a list of address tags under the hood, constructed by tagging address components using `addr_tag()` and combining them into specific fields:

- `street_number`: `AddressNumber`
- `street_name`: `StreetNamePreType`, `StreetNamePreDirectional`, `StreetName`
- `street_type`: `StreetNamePostType`, `StreetNamePostDirectional`
- `city`: `PlaceName`
- `state`: `StateName`
- `zip_code`: `ZipCode`

Usage

```
addr(
  x = character(),
  clean_address_text = TRUE,
  expand_street_type = TRUE,
  abbrev_cardinal_dir = TRUE,
  clean_zip_code = TRUE
)

as_addr(x, ...)
```

Arguments

`x`	a character vector of address strings
`clean_address_text`	logical; use `clean_address_text()` to clean address text prior to tagging?
`expand_street_type`	logical; use `expand_post_type()` to expand `StreetNamePostType` tags? (e.g., "Ave" -> "Avenue")
`abbrev_cardinal_dir`	logical; abbreviate cardinal directions? (e.g., "west" -> "w")
`clean_zip_code`	logical; remove any non-digit (or hyphen) characters and truncate tagged ZIP Code to 5 characters?
`...`	used to pass arguments in `as_addr` to underlying `addr()`

Details

In addition to the cleaning steps described in the arguments, the street number is coerced to a numeric after removing non-numeric characters. See `addr_tag()` for details on address component tagging.

In the case of an address having more than one word for a tag (e.g., "Riva Ridge" for `StreetName`), then these are concatenated together, separated by a space in the order they appeared in the address.

Compared to using `addr()`, `as_addr()` processes input character strings such that parsing is done once per unique input, usually speeding up address parsing in real-world datasets where address strings are often duplicated across observations.

Examples

```
as_addr(c("3333 Burnet Ave Cincinnati OH 45229", "1324 Burnet Ave Cincinnati OH 45229"))
```

addr_match *matching addr vectors*

Description

For an addr vector, the string distances are calculated between a reference addr vector (`ref_addr`). A list of matching reference addr vectors less than or equal to the specified optimal string alignment distances are returned. See `stringdist::stringdist-metrics` for more details on string metrics and the optimal string alignment (`osa`) method.

Usage

```
addr_match(
  x,
  ref_addr,
  stringdist_match = c("osa_lt_1", "exact"),
  match_street_type = TRUE,
  simplify = TRUE
)

addr_match_street_name_and_number(
  x,
  ref_addr,
  stringdist_match = c("osa_lt_1", "exact"),
  match_street_type = TRUE,
  simplify = TRUE
)

addr_match_street(
  x,
  ref_addr,
  stringdist_match = c("osa_lt_1", "exact"),
  match_street_type = TRUE
)
```

Arguments

`x`	an addr vector to match
`ref_addr`	an addr vector to search for matches in
`stringdist_match`	method for determining string match of street name: "osa_lt_1" requires an optimized string distance less than 1; "exact" requires an exact match
`match_street_type`	logical; require street type to be identical to match?
`simplify`	logical; randomly select one addr from multi-matches and return an addr() vector instead of a list? (empty addr vectors and NULL values are converted to NA)

Value

for `addr_match()` and `addr_match_street_name_number()`, a named list of possible addr matches for each addr in `x`

for addr_match_street, a list of possible addr matches for each addr in `x` (as `ref_addr` indices)

Examples

```
addr(c("3333 Burnet Ave Cincinnati OH 45229", "5130 RAPID RUN RD CINCINNATI OHIO 45238")) |>
  addr_match(cagis_addr()$cagis_addr)

addr(c("3333 Burnet Ave Cincinnati OH 45229", "5130 RAPID RUN RD CINCINNATI OHIO 45238")) |>
  addr_match(cagis_addr()$cagis_addr, simplify = FALSE) |>
  tibble::enframe(name = "input_addr", value = "ca") |>
  dplyr::mutate(ca = purrr::list_c(ca)) |>
  dplyr::left_join(cagis_addr(), by = c("ca" = "cagis_addr")) |>
  tidyr::unnest(cols = c(cagis_addr_data)) |>
  dplyr::select(-ca, -cagis_address)
```

`addr_match_geocode` *Geocode addr vectors*

Description

Addresses are attempted to be matched to reference geographies using different methods associated with decreasing levels of precision in the order listed below. Each method generates matched s2 cell identifiers differently and is recorded in the `match_method` column of the returned tibble:

1. `ref_addr`: reference s2 cell from direct match to reference address
2. `tiger_range`: centroid of street-matched TIGER address ranges containing street number
3. `tiger_street`: centroid of street-matched TIGER address ranges closest to the street number
4. `none`: unmatched using all previous approaches; return missing s2 cell identifier

Usage

```
addr_match_geocode(
  x,
  ref_addr = cagis_addr()$cagis_addr,
  ref_s2,
  county = "39061",
  year = "2022"
)
```

Arguments

`x`	an addr vector (or character vector of address strings) to geocode
`ref_addr`	an addr vector to search for matches in
`ref_s2`	a s2_cell vector of locations for each ref_addr
`county`	character county identifer for TIGER street range files to search for matches in
`year`	character year for TIGER street range files to search for matches in

Details

Performance was compared to the degauss geocoder (see `/inst/compare_geocoding_to_degauss.R`) using real-world addresses in `voter_addresses()`. Match success rates were similar, but DeGAUSS matched about 5% more of the addresses. These differences are sensitive to the match criteria considered for DeGAUSS (here precision of 'range' & score > 0.7 *or* precision of 'street' & score > 0.55):

addr_matched	degauss_matched	n	perc
TRUE	TRUE	224714	92.8%
FALSE	TRUE	13407	5.5%
FALSE	FALSE	2993	1.2%
TRUE	FALSE	1019	0.4%

Among those that were geocoded by both, 97.7% were geocoded to the same census tract, and 96.6% to the same block group:

ct_agree	bg_agree	n	s2_dist_ptiles (5th, 25th, 50th, 75th, 95th)	perc
TRUE	TRUE	217179	14.7, 24.3, 39, 68.9, 153.6	96.6%
FALSE	FALSE	4805	21.6, 39.2, 158.9, 5577.9, 16998.8	2.1%
TRUE	FALSE	2730	19.6, 28.6, 41.2, 94.8, 571.8	1.2%

Value

a tibble with columns: `addr` contains `x` converted to an `addr` vector, `s2` contains the resulting geocoded s2 cells as an `s2cell` vector, `match_method` is a factor with levels described above

Examples

```
set.seed(1)
cagis_s2 <-
  cagis_addr()$cagis_addr_data |>
  purrr::modify_if(\(.) length(.) > 0 && nrow(.) > 1, dplyr::slice_sample, n = 1) |>
  purrr::map_vec(purrr::pluck, "cagis_s2", .default = NA, .ptype = s2::s2_cell())
addr_match_geocode(x = sample(voter_addresses(), 100), ref_s2 = cagis_s2) |>
  print(n = 100)
```

addr_match_tiger_street_ranges

Match an addr vector to TIGER street ranges

Description

Match an addr vector to TIGER street ranges

Usage

```
addr_match_tiger_street_ranges(
  x,
  county = "39061",
  year = "2022",
  street_only_match = c("none", "all", "closest"),
  summarize = c("none", "union", "centroid")
)
```

Arguments

`x`	an addr vector to match
`county`	character string of county identifier
`year`	year of tigris product
`street_only_match`	for addresses that match a TIGER street name, but have street numbers that don't intersect with ranges of potential street numbers, return `"none"`, `"all"`, or the `"closest"` range geographies
`summarize`	optionally summarize matched street ranges as their union or centroid

Details

To best parse street names and types, this function appends dummy address components just for the purposes of matching tiger street range names (e.g., `1234 {tiger_street_name} Anytown AB 00000`)

Value

a list of matched tigris street range tibbles; a NULL value indicates that no street name was matched; if `street_only_match` is FALSE, a street range tibble with zero rows indicates that although a street was matched, there was no range containing the street number

Examples

```
my_addr <- as_addr(c("224 Woolper Ave", "3333 Burnet Ave", "33333 Burnet Ave", "609 Walnut St"))

addr_match_tiger_street_ranges(my_addr, county = "39061", street_only_match = "all")

addr_match_tiger_street_ranges(my_addr, county = "39061", summarize = "centroid")
addr_match_tiger_street_ranges(my_addr, county = "39061",
                               street_only_match = "closest", summarize = "centroid") |>
  dplyr::bind_rows() |>
  dplyr::mutate(census_bg_id = s2_join_tiger_bg(s2::as_s2_cell(s2_geography)))
```

`addr_tag` *Tag components of an address string*

Description

The address components are tagged using a rust port of usaddress. Component names are based upon the United States Thoroughfare, Landmark, and Postal Address Data Standard.

Usage

```
addr_tag(x, clean_address_text = TRUE)
```

Arguments

`x`	a character vector of addresses
`clean_address_text`	logical; use clean_address_text() to clean addresses prior to tagging?

Details

Possible address labels include:

- `AddressNumberPrefix`
- `AddressNumberSuffix`
- `AddressNumber`
- `BuildingName`
- `CornerOf`
- `IntersectionSeparator`
- `LandmarkName`
- `NotAddress`
- `OccupancyIdentifier`
- `OccupancyType`
- `PlaceName`
- `Recipient`
- `StateName`
- `StreetNamePostDirectional`
- `StreetNamePostType`

- `StreetNamePreDirectional`
- `StreetNamePreModifier`
- `StreetNamePreType`
- `StreetName`
- `SubaddressIdentifier`
- `SubaddressType`
- `USPSBoxGroupID`
- `USPSBoxGroupType`
- `USPSBoxID`
- `USPSBoxType`
- `ZipCode`

Find more information about the definitions here

Value

a list, the same length as x, of named character vectors of address component tags; each vector contains all space-separated elements of the cleaned address and are each named based on inferred address labels (see Details)

Examples

```
addr_tag(c("290 Ludlow Avenue Apt #2 Cincinnati OH 45220", "3333 Burnet Ave Cincinnati OH 45219"))
```

`cagis_addr` *CAGIS Addresses*

Description

CAGIS Addresses

Usage

```
cagis_addr()
```

Value

An example tibble created from the CAGIS addresses with a pre-calculated, unique `cagis_addr` vector column. The `cagis_addr_data` column is a list of tibbles because one CAGIS address can correspond to multiple parcel identifiers and address-level data (place, type, s2, etc.). See `inst/make_cagis_addr.R` for source code to create data, including filtering criteria:

- use only addresses that have `STATUS` of `ASSIGNED` or `USING` and are not orphaned (`ORPHANFLG == "N"`)
- omit addresses with `ADDRTYPE`s that are milemarkers (`MM`), parks (`PAR`), infrastructure projects (`PRJ`), cell towers (`CTW`), vacant or commercial lots (`LOT`), and other miscellaneous non-residential addresses (`MIS`, `RR`, `TBA`)
- s2 cell is derived from LONGITUDE and LATITUDE fields in CAGIS address database

Examples

```
cagis_addr()
```

`clean_address_text` *clean address text*

Description

remove excess whitespace; keep only letters, numbers, and `-`

Usage

```
clean_address_text(.x)
```

Arguments

`.x`	a vector of address character strings

Value

a vector of cleaned addresses

Examples

```
clean_address_text(c(
  "3333 Burnet Ave Cincinnati OH 45219",
  "33_33 Burnet Ave. Cincinnati OH 45219",
  "33\\33 B\"urnet Ave; Ci!ncinn&*ati OH 45219",
  "3333 Burnet Ave Cincinnati OH 45219",
  "33_33 Burnet Ave. Cincinnati OH 45219"
))
```

`elh_data` *Example real-world data with line-one-only addresses*

Description

The Cincinnati Evicition Hotspots data was downloaded from Eviction Labs and contains characteristics of the top 100 buildings that are responsible for about 25% of all eviction filings in Cincinnati (from their "current through 8-31-2024" release).

Usage

```
elh_data()
```

Details

https://evictionlab.org/eviction-tracking/cincinnati-oh/

Value

a tibble with 100 rows and 9 columns

Examples

```
elh_data()
```

`expand_post_type` *Expand street name post type*

Description

Abbreviations of street type (e.g., "Ave", "St") are converted to expanded versions (e.g., "Avenue", "Street").

Usage

```
expand_post_type(x)
```

Arguments

`x`	character vector of `StreetnamePostType` abbreviations

Value

a character vector of the same length containing the expanded street name post type

Examples

```
expand_post_type(c("ave", "av", "Avenue", "tl"))
```

`get_tiger_block_groups` *get s2_geography for census block groups*

Description

get s2_geography for census block groups

Usage

```
get_tiger_block_groups(state, year)
```

Arguments

`state`	census FIPS state identifier
`year`	vintage of TIGER/Line block group geography files

Value

a tibble with `GEOID` and `s2_geography` columns

Examples

```
get_tiger_block_groups(state = "39", year = "2022")
```

`get_tiger_street_ranges` *Get tigris street range geography files from census.gov*

Description

Downloaded files are cached in `tools::R_user_dir("addr", "cache")`. Street ranges with missing minimum or maximum address numbers are excluded.

Usage

```
get_tiger_street_ranges(county, year = "2022")
```

Arguments

`county`	character string of county identifier
`year`	year of tigris product

Value

a list of tibbles, one for each street name, with `TLID`, `s2_geography`, `from`, and `to` columns

Examples

```
Sys.setenv("R_USER_CACHE_DIR" = tempfile())
get_tiger_street_ranges("39061")[1001:1004]
```

`s2_join_tiger_bg` *Tiger Block Groups*

Description

Get the identifier of the closest census block group based on the intersection of the s2 cell locations with the the US Census TIGER/Line shapefiles

Usage

```
s2_join_tiger_bg(x, year = as.character(2013:2023))
```

Arguments

`x`	s2_cell vector
`year`	vintage of TIGER/Line block group geography files

Value

character vector of matched census block group identifiers

Examples

```
s2_join_tiger_bg(x = s2::as_s2_cell(c("8841b39a7c46e25f", "8841a45555555555")), year = "2023")
```

`tiger_states` *get s2_geography for census states*

Description

get s2_geography for census states

Usage

```
tiger_states(year)
```

Arguments

`year`	vintage of TIGER/Line block group geography files

Value

a tibble with `GEOID` and `s2_geography` columns

Examples

```
tiger_states(year = "2022")
```

`usaddress_tag` *Return list of lists of address tags to R.*

Description

Return list of lists of address tags to R.

Usage

```
usaddress_tag(input)
```

Arguments

`input`	character string of addresses

`voter_addresses` *Example real-world addresses*

Description

The voter_addresses data was generated as an example character vector of real-world addresses. These addresses were downloaded from the Hamilton County, Ohio voter registration database on 2024-09-12. See `inst/make_example_addresses.R` for more details. `AddressPreDirectional`, `AddressNumber`, `AddressStreet`, `AddressSuffix`, `CityName`, "OH", and `AddressZip` are pasted together to create 242,133 unique addresses of registered voters in Hamilton County, OH.

Usage

```
voter_addresses()
```

Value

a character vector

Examples

```
voter_addresses() |>
  head()
```

Index