Package: locatr (via r-universe)

July 23, 2026

Title Audit-Ready Geocoding and Local Geography for Messy Location Data

Version 0.1.0

Description Cleans, geocodes, validates, reviews, and exports messy location address data supplied by the user. The package sits on top of 'tidygeocoder': it calls geocoding services, rejects implausible coordinates with configurable region guards, applies fallback name/address matching, joins points to optional local geography with 'sf', and records an audit trail showing how each coordinate was produced. Outputs are designed for manual review, dashboards, and reusable location crosswalks.

License MIT + file LICENSE

URL https://prigasg.github.io/locatr/, https://github.com/PrigasG/locatr

BugReports https://github.com/PrigasG/locatr/issues

Depends R (>= 4.1.0)

Imports datasets, dplyr (>= 1.1.0), magrittr, readr, rlang (>= 1.1.0), sf, stats, stringr, tibble, tidygeocoder, utils

Suggests arrow, bslib, DT, httr, jsonlite, knitr, leaflet, pkgdown, readxl, rmarkdown, shiny, spelling, testthat (>= 3.1.7), tigris, writexl

VignetteBuilder knitr

Config/Needs/website pkgdown

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

Language en-US

NeedsCompilation no

Author George Arthur [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-1975-1459>)

Maintainer George Arthur <prigasgenthian48@gmail.com>

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

Date/Publication 2026-07-23 14:20:08 UTC

RemoteUrl https://github.com/cran/locatr

RemoteRef HEAD

RemoteSha b4cb7845d8d4c0319dd635aae393ff784caaa598

Contents

add_census_geographies *Attach multiple Census geography levels to geocoded points*

Description

Enriches geocoded records with one or more Census TIGER/Line geography levels in a single call, by point-in-polygon assignment. Where add_county_muni() answers "which county and municipality", this answers "which tract, block group, ZCTA, congressional district, state legislative district, or school district" - the analysis and policy geographies that dashboards and grant reporting often need. Each requested level adds two columns: <level>_geoid (the Census GEOID) and <level>_name (the layer's name field, NA where the layer has none, e.g. ZCTAs).

Usage

```
add_census_geographies(
  data,
  state,
  levels = "tract",
  county = NULL,
  year = NULL,
  cb = TRUE,
  ...
)
```

Arguments

data	A geocoded data frame with latitude and longitude columns.
state	Two-letter state abbreviation or FIPS code (passed to **tigris**). ZCTAs are national, so state is ignored for the "zcta" level.
levels	Character vector of geography levels to attach. Any of "tract", "block_group", "zcta", "county", "place", "county_subdivision", "congressional_district", "state_legislative_district_upper", "state_legislative_district_lower", "school_district" (unified). Defaults to "tract".
county	Optional county filter (name or FIPS) for the levels that accept one ("tract", "block_group", "county_subdivision").
year	Vintage year for the boundary files. NULL uses the **tigris** default.
cb	If TRUE (default), use the smaller cartographic boundary files.
...	Passed through to the underlying **tigris** download functions.

Details

The geography step only needs latitude/longitude; it does not touch the address columns. Downloads use **tigris** and therefore need network access.

Value

`data` with, for each requested level, a `<level>_geoid` and `<level>_name` column. Rows without usable coordinates get `NA`.

See Also

`add_county_muni()` for county/municipality, `build_local_geography()` for a single reusable boundary layer.

Examples

```
if (interactive()) {
  enriched <- add_census_geographies(
    geocoded, state = "NJ",
    levels = c("tract", "congressional_district", "school_district")
  )
}
```

---

add_county_muni *Add county and municipality fields from Census boundaries*

Description

Convenience wrapper for the common workflow where users want county and municipality/locality fields but do not already have a boundary file. It builds a Census TIGER/Line geography layer with `build_local_geography()` and then applies `add_muni_from_shapes()`.

Usage

```
add_county_muni(
  data,
  state,
  geography = c("county_subdivision", "place", "county", "tract"),
  county = NULL,
  year = NULL,
  cb = TRUE,
  ...
)
```

Arguments

`data`	A validated data frame with `latitude`/`longitude`.
`state`	Two-letter state abbreviation or FIPS code.
`geography`	Which Census layer should become `Municipality`; passed to `build_local_geography()`.
`county`	Optional county filter for supported Census layers.
`year`	Vintage year for the boundary files.
`cb`	If `TRUE`, use smaller cartographic boundary files.
`...`	Passed through to `build_local_geography()`.

Details

For reporting where municipality has a state-specific legal definition, use `add_muni_from_shapes()` with an official state/local GIS layer.

Value

`data` with `County`, `Municipality`, `Muni Key`, stable code/join columns when Census provides them, and `muni_match_status`, plus the generic `location_*` geography audit fields.

`add_local_geography` *Join records to local geography*

Description

Spatially joins geocoded points to a local polygon layer and returns selected geography attributes for dashboards. County and locality column names are auto-detected from common boundary schemas, or can be set explicitly.

Usage

```
add_local_geography(
  data,
  geography_shapes = NULL,
  county_col = NULL,
  locality_col = NULL
)
```

Arguments

`data`	A validated data frame with `latitude`/`longitude`.
`geography_shapes`	An `sf` polygon layer, or `NULL` to use packaged data.
`county_col, locality_col`	Optional explicit column names in `geography_shapes`. When `NULL`, `add_local_geography()` guesses from common names, preferring `location_county`/`location_locality` when present.

Details

If `geography_shapes` is `NULL`, the function looks for a packaged `local_geography` dataset (for production this is the NJGIN/NJOGIS municipal boundary layer built by `data-raw/local_geography.R`, whose attributes are already named `location_county`/`location_locality`). Pass an `sf` polygon layer to adapt this join to another state, county, or service area.

For NJ production maps, `location_locality` is taken from an authoritative municipal boundary polygon - not from the geocoder response or Census reverse-geocoding, whose "county subdivision" names only look municipal - so every locality is traceable to a named boundary source.

Value

data with location_county, location_locality, and geography_match_status. Rows without usable coordinates are kept (audit-safe) with NA geography.

---

add_match_confidence *Add a unified match-confidence score*

Description

Collapses locatr's several trust signals into one calibrated match_confidence on a 0-1 scale plus a short confidence_reason string, so a reviewer can sort or threshold on a single column instead of reading match_status, validation_status, nm_status, and review_status together. Higher is more trustworthy.

Usage

```
add_match_confidence(data)
```

Arguments

data	A data frame from the batch pipeline or from geocode_address().

Details

The right scoring model is chosen from the columns present:

- Pipeline output (from geocode_records() / the crosswalk): scored from the tier that placed the row (geocode_pass), the match and validation status, and the name-tier confidence. Rejected or unplaced rows score near zero; reference-verified and manual rows score highest.
- Candidate output (from geocode_address()): scored from the ArcGIS match score, discounted by how coarse the address type is, and capped when the point falls outside a supplied bbox.

The score is a transparent, rule-based prior - deliberately explainable rather than a black-box model - so every value can be traced to its confidence_reason.

Value

data with two added columns: match_confidence (0-1, rounded to three decimals) and confidence_reason.

Examples

```
add_match_confidence(data.frame(
  geocode_pass = c("pass_1_census_structured", "pass_4_name_lookup"),
  match_status = c("matched", "matched_low_confidence"),
  validation_status = c("coordinate_ok", "coordinate_ok"),
  latitude = c(40.2, 40.3), longitude = c(-74.7, -74.8)
))
```

`add_muni_from_key` *Add county and municipality fields by joining on a shared key*

Description

The non-spatial counterpart to `add_muni_from_shapes()`. Instead of a point-in-polygon join, it merges geography attributes from a boundary layer onto geocoded records by a shared key column (for example a ZIP, FIPS, GEOID, or local region code that both tables carry). Use it when a spatial join is not the right criterion - the records may lack coordinates, or the authoritative geography is keyed by a code rather than a polygon footprint.

Usage

```
add_muni_from_key(
  data,
  muni_shapes,
  data_key,
  shp_key,
  county_col = NULL,
  muni_col = NULL,
  key_col = NULL
)
```

Arguments

`data`	A data frame of geocoded records carrying `data_key`.
`muni_shapes`	An `sf` polygon (or attribute) layer carrying `shp_key` and the geography attributes to attach.
`data_key`	Name (string) of the join-key column in `data`.
`shp_key`	Name (string) of the join-key column in `muni_shapes`.
`county_col, muni_col`	Optional explicit county / locality column names in `muni_shapes`. Empty strings are treated as unset.
`key_col`	Optional municipal key column in `muni_shapes`.

Details

Output columns match `add_muni_from_shapes()` exactly, so the two join paths are interchangeable downstream: `County`, `Municipality`, `Muni Key`, `muni_join_key`, the stable code/identifier fields, `location_county`, `location_locality`, `geography_match_status`, and `muni_match_status`. Stable code fields (`county_code`, `municipality_code`, `municipality_geoid`, etc.) are auto-detected from common boundary schemas; `county_fips` is synthesised from a state FIPS plus county code when not supplied directly, and `Muni Key` falls back to `County::Municipality` when no official identifier exists.

Value

data with County, Municipality, Muni Key, muni_join_key, county_code, county_fips, municipality_code, municipality_geoid, municipality_name_standard, municipality_type, muni_match_status, and the generic location_county / location_locality / geography_match_status audit fields.

See Also

add_muni_from_shapes() for the spatial (point-in-polygon) join.

---

add_muni_from_shapes *Add county and municipality fields from boundary polygons*

Description

A crosswalk-oriented wrapper around add_local_geography() for workflows where the final output should carry explicit county/municipality columns and stable join identifiers. It spatially joins geocoded points to municipal/local boundary polygons and adds County, Municipality, Muni Key, muni_join_key, code fields, and muni_match_status.

Usage

```
add_muni_from_shapes(
  data,
  muni_shapes,
  county_col = NULL,
  muni_col = NULL,
  key_col = NULL
)
```

Arguments

data	A validated data frame with latitude/longitude.
muni_shapes	An sf polygon layer containing county and municipality attributes.
county_col, muni_col	Optional explicit column names in muni_shapes. When NULL, common names are auto-detected.
key_col	Optional municipal key column in muni_shapes.

Details

muni_join_key is copied from key_col when supplied, or auto-detected from common stable identifier columns such as GEOID and MUNI_KEY. Muni Key is retained as a readable key and falls back to County::Municipality when no official identifier exists.

Value

data with County, Municipality, Muni Key, muni_join_key, county_code, county_fips, municipality_code, municipality_geoid, municipality_name_standard, municipality_type, and muni_match_status when available. The generic location_county, location_locality, and geography_match_status columns are retained.

---

apply_manual_overrides

Apply completed manual overrides

Description

Joins a reviewer-completed override file (same layout write_geocode_review() produced) and coalesces verified coordinates and geography over the automated values. Overrides are themselves bbox-checked so a typo can't drop a point in the ocean.

Usage

```
apply_manual_overrides(data, override_file, bbox = region_bbox("NJ"))
```

Arguments

data	A data frame with record_id and the audit columns.
override_file	Path to the completed override CSV.
bbox	Bounding box for validating manual coordinates; see region_bbox().

Value

data with overrides applied and manual_override_used set.

---

backfill_from_reference

Backfill verified coordinates from a trusted reference table (Tier 0)

Description

The authoritative first tier of the cascade. Joins coordinates from a curated key -> coordinates table - an institutional-memory table of records whose location was resolved and checked at some point - over the automated geocoders, so a record that has already been verified never has to be re-geocoded. This is what turns one analyst's manual review into a permanent asset: feed last cycle's completed overrides (or any trusted coordinate list) back in as reference, and those rows are placed instantly and exactly.

Usage

```
backfill_from_reference(
  data,
  reference,
  by = "record_id",
  lat_col = "latitude",
  lon_col = "longitude",
  county_col = NULL,
  locality_col = NULL,
  bbox = region_bbox("NJ")
)
```

Arguments

data	A data frame from flag_bad_addresses() (or any frame carrying the key column).
reference	A data frame of verified records: the key column plus coordinate columns. May also carry county/locality columns. NULL or an empty frame makes this a no-op.
by	Name of the key column shared by data and reference (default "record_id").
lat_col, lon_col	Coordinate column names in reference (default "latitude"/"longitude").
county_col, locality_col	Optional geography column names in reference to backfill into location_county/location_locality
bbox	Bounding box used to reject out-of-region reference coordinates; see region_bbox().

Details

Reference coordinates are still bbox-validated, so a stale or fat-fingered entry cannot drop a point outside the region. Because the reference is authoritative, a matched row is marked review_status == "reference_backfilled" and carries valid coordinates, so geocode_census() and every later tier skip it automatically - even if its raw address was previously flagged (e.g. a PO box whose true coordinates were verified once).

Run this before geocode_census() (the cascade does so when you pass reference = to geocode_records()).

Value

data with reference audit columns ref_latitude, ref_longitude, ref_status, and, for rows the reference filled, updated latitude/longitude/geocode_method/geocode_pass ("pass_0_reference")/ match_status/review_status.

Examples

```
records <- tibble::tibble(
  record_id = c("a", "b"),
  review_status = c("ready_for_geocoding", "needs_manual_review")
)
verified <- tibble::tibble(record_id = "b", latitude = 40.22, longitude = -74.76)
backfill_from_reference(records, verified)
```

bbox_from_sf *Build a geocoding bounding box from an sf layer*

Description

Converts any point, line, or polygon `sf` layer to WGS84 and returns a named latitude/longitude bounding box suitable for `geocode_records()`, `geocode_arcgis()`, `geocode_by_name()`, and `validate_geocodes()`. This is the safest way to keep multi-state geocoding and local geography joins aligned: build or load the geography layer first, then derive the bbox from it.

Usage

```
bbox_from_sf(geography_shapes, buffer = 0.05)
```

Arguments

`geography_shapes`	An `sf` object.
`buffer`	Numeric buffer in decimal degrees added to each side of the bounding box. Defaults to `0.05` to avoid rejecting edge locations.

Value

A named numeric vector with `lat_min`, `lat_max`, `lon_min`, and `lon_max`.

Examples

```
if (interactive()) {
areas <- build_local_geography("PA")
bbox <- bbox_from_sf(areas)
}
```

build_local_geography *Build a local geography layer from Census TIGER/Line boundaries*

Description

Downloads an authoritative Census boundary layer for a state with the **tigris** package and standardises it into the two-column schema `add_local_geography()` expects: `location_county` (always from counties) and `location_locality` (from the requested geography). It also carries stable Census identifiers such as `county_fips`, `municipality_geoid`, and `muni_join_key` when those fields exist. This makes "locality" a configurable concept, because what counts as a municipality is not consistent across states.

Usage

```
build_local_geography(
  state,
  geography = c("county_subdivision", "place", "county", "tract"),
  county = NULL,
  year = NULL,
  cb = TRUE,
  ...
)
```

Arguments

state	Two-letter state abbreviation or FIPS code (passed to **tigris**).
geography	Which Census layer becomes location_locality. One of "county_subdivision", "place", "county", "tract".
county	Optional county filter (name or FIPS) for "county_subdivision"/"tract"; passed to **tigris**.
year	Vintage year for the boundary files. NULL uses the **tigris** default.
cb	If TRUE (default), use the smaller cartographic boundary files; FALSE pulls the full-resolution TIGER/Line files.
...	Passed through to the underlying **tigris** download function.

Details

Choosing geography:

- "county_subdivision" (default) maps well to townships/municipalities in states like NJ, PA, NY and New England. Best general default for "locality".
- "place" maps to incorporated places and CDPs, but misses townships and many unincorporated areas. Places can also straddle counties, so each place is assigned the county it overlaps most.
- "county" sets locality to the county itself.
- "tract" uses the census tract identifier as the locality (analysis, not administrative naming).

Census TIGER/Line is the best scalable baseline nationwide. For high-stakes, state-specific reporting where "municipality" has legal meaning, prefer an official state GIS layer and pass it straight to add_local_geography() - that path works regardless, since the join accepts any polygon layer.

Value

An sf polygon layer in WGS84 (EPSG:4326) with location_county, location_locality, stable join-code columns when available, and geometry, ready for add_local_geography(geography_shapes = ...).

Examples

```
if (interactive()) {
areas <- build_local_geography(state = "PA", geography = "county_subdivision")
final <- add_local_geography(geocoded, geography_shapes = areas)
}
```

`build_review_overrides` *Build a manual-override table from review decisions*

Description

Turns a set of per-record review decisions (accept / reject / relocate) into a completed override table with the same layout `write_geocode_review()` produces, so it can be written to CSV and fed straight to `apply_manual_overrides()`. This is the testable core behind the map-based review app (`run_locatr_review_app()`); it holds no reactive or UI code, so it can also be used headless.

Usage

```
build_review_overrides(data, decisions)
```

Arguments

`data`	A geocoded data frame with at least `record_id`, and ideally `latitude`/`longitude`/`record_name`/`full_ad` for context.
`decisions`	A data frame with `record_id`, `decision` (one of `"accept"`, `"reject"`, `"relocate"`), and - for relocations - `manual_latitude` / `manual_longitude`.

Details

Decision semantics:

- `accept` - confirm the automated coordinate: `manual_latitude` / `manual_longitude` are set to the record's current coordinate.
- `relocate` - use the reviewer's coordinate: `manual_latitude` / `manual_longitude` come from the `decisions` table.
- `reject` - drop the coordinate: `manual_*` are left `NA` (so `apply_manual_overrides()` does not place it) and the note records the rejection.

Value

A tibble with `record_id`, `record_name`, `full_address_clean`, the current `latitude`/`longitude`, the reviewer's `manual_latitude` / `manual_longitude`, and `manual_note`. Compatible with `apply_manual_overrides()`.

See Also

`run_locatr_review_app()`, `apply_manual_overrides()`, `write_geocode_review()`

Examples

```
geocoded <- data.frame(
  record_id = c("a", "b", "c"),
  record_name = c("A", "B", "C"),
  full_address_clean = c("1 A St", "2 B St", "3 C St"),
  latitude = c(40.1, 40.2, NA),
  longitude = c(-74.1, -74.2, NA)
)
decisions <- data.frame(
  record_id = c("a", "b", "c"),
  decision = c("accept", "relocate", "reject"),
  manual_latitude = c(NA, 40.25, NA),
  manual_longitude = c(NA, -74.25, NA)
)
build_review_overrides(geocoded, decisions)
```

`cache_clear` *Clear a locatr cache*

Description

Empties the in-memory table. For a persistent cache this also deletes the file, so it is guarded: a persistent cache requires `confirm = TRUE`.

Usage

```
cache_clear(cache, confirm = FALSE)
```

Arguments

`cache`	A `locatr_cache` from `locatr_cache()`.
`confirm`	Must be `TRUE` to clear a persistent (path-backed) cache and delete its file. Ignored for memory-only caches.

Value

The cleared `cache`, invisibly.

See Also

`locatr_cache()`

cache_info *Summarise a locatr cache*

Description

Summarise a locatr cache

Usage

```
cache_info(cache)
```

Arguments

cache	A locatr_cache from locatr_cache().

Value

A one-row tibble: rows, distinct keys, distinct methods, oldest/newest cached_at, whether it is persistent, its path, and format.

See Also

locatr_cache()

clean_addresses *Clean and standardise address fields*

Description

Normalises raw address, city and ZIP text into geocoder-friendly columns and builds a single-line full_address_clean. Column mappings are supplied with tidy-eval (bare column names). Original address pieces are preserved in *_raw columns. If the input already contains a full_address_clean column in any case style (for example Full_Address_Clean), locatr preserves that user-supplied value as full_address_raw so there is only one canonical full_address_clean column after cleaning.

Usage

```
clean_addresses(
  data,
  id = NULL,
  address,
  city,
  zip = NULL,
  name = NULL,
  state = "NJ"
)
```

Arguments

data	A data frame of records with addresses.
id	Optional bare column name holding a unique record identifier. When omitted, record_id is generated from the row number.
address, city	Bare column names for the raw address and city. Required.
zip	Optional bare column name for the raw ZIP/postal code. When omitted, zip_clean is NA.
name	Optional bare column name for the record name (kept as record_name for review/exports). Defaults to NULL.
state	Two-letter state used for all rows. Defaults to "NJ" for the first production workflow; pass another state abbreviation as needed.

Details

Only address and city are required. When id is omitted, a surrogate record_id is generated from the row position. When zip is omitted (or empty), zip_clean is NA and full_address_clean is built without a trailing ZIP, so an address + city + state row is still geocodable. Supplying a ZIP improves Census structured-match precision but is no longer mandatory.

Value

data with added columns: record_id, record_name, address_raw, city_raw, zip_raw, optional full_address_raw, address_clean, city_clean, state_clean, zip_clean, full_address_clean.

Examples

```
df <- tibble::tibble(
  LocationID = "NJ306100", Name = "Hackensack-UMC Mountainside",
  Address = "ONE BAY AVE", City = "Montclair", Zip = "7042"
)
clean_addresses(df, id = LocationID, address = Address,
                      city = City, zip = Zip, name = Name)

# address + city only (surrogate id, no ZIP)
clean_addresses(tibble::tibble(Address = "100 Main St", City = "Trenton"),
                address = Address, city = City)
```

---

compare_geocode_runs *Compare two geocoding runs*

Description

Finds records whose coordinates, review status, geocode pass, or geography assignment changed between two runs. This is useful after changing thresholds, adding a reference file, or swapping geography sources.

Usage

```
compare_geocode_runs(
  old,
  new,
  by = "record_id",
  coordinate_tolerance = 1e-06,
  changed_only = TRUE
)
```

Arguments

old	Previous locatr output.
new	New locatr output.
by	Key column used to match rows. Defaults to record_id.
coordinate_tolerance	Numeric tolerance for latitude/longitude changes.
changed_only	If TRUE (default), return only rows with at least one tracked change.

Value

A tibble with old/new values and change flags.

Examples

```
old <- tibble::tibble(record_id = "a", latitude = 40, longitude = -75)
new <- tibble::tibble(record_id = "a", latitude = 41, longitude = -75)
compare_geocode_runs(old, new)
```

---

explain_geocode_result

Explain how geocoded records were handled

Description

Turns the main audit columns into short, reviewer-friendly sentences. This is useful when checking a few records by hand or when adding plain-English notes to a review export.

Usage

```
explain_geocode_result(data, row = NULL)
```

Arguments

data	A locatr output data frame.
row	Optional row selector. Use NULL for all rows (default), a numeric row index, or a record_id value.

Value

A character vector of explanations.

Examples

```
x <- tibble::tibble(
  record_id = "a",
  geocode_pass = "pass_4_name_lookup",
  match_status = "matched_low_confidence",
  validation_status = "coordinate_ok",
  review_status = "needs_manual_review",
  nm_score = 87,
  nm_addr_type = "POI"
)
explain_geocode_result(x)
```

---

export_location_crosswalk

Export the location crosswalk

Description

Selects the final, stable set of columns for dashboards, GIS joins, and reusable reference tables, and optionally writes them to CSV. Audit columns are retained so a reviewer can always see how each coordinate was produced, including score/type/status fields from the name lookup tier when available.

Usage

```
export_location_crosswalk(data, path = NULL)
```

Arguments

`data`	A fully processed data frame.
`path`	Optional output CSV path. When `NULL`, nothing is written.

Value

The crosswalk tibble (also written to `path` when supplied).

`flag_bad_addresses` *Flag addresses that should not be blindly geocoded*

Description

Identifies PO boxes, placeholders, and missing fields so they go straight to review instead of wasting geocoder calls (or producing confident-but-wrong matches). Sets `bad_address_flag` and an initial `review_status`.

Usage

```
flag_bad_addresses(data)
```

Arguments

`data`	A data frame from `clean_addresses()`.

Details

A missing ZIP is recorded as `bad_address_flag == "missing_zip"` for audit, but it does **not** block geocoding: as long as the address and city are present, the row stays `ready_for_geocoding` (Census matches on street/city/state and ArcGIS on the single-line address). Only genuinely unusable rows - missing address or city, PO boxes, placeholders, test records - are routed to `needs_manual_review`.

Value

`data` with added columns `bad_address_flag` and `review_status`. Rows fit for geocoding get `review_status == "ready_for_geocoding"`.

Examples

```
df <- tibble::tibble(
  record_id = c("a", "b"),
  address_clean = c("100 MAIN STREET", "PO BOX 42"),
  city_clean = c("TRENTON", "TRENTON"),
  zip_clean = c("08608", "08608"),
  record_name = c("Real Site", "Mailbox Co")
)
flag_bad_addresses(df)
```

`flag_field_conflicts` *Flag cross-field conflicts in location data*

Description

Catches a class of data-entry errors the geocoder itself will silently accept: a ZIP that cannot belong to the stated state, and a stated county that disagrees with the county the coordinate actually fell in. It adds audit columns rather than changing any coordinate, so a reviewer can decide what to do.

Usage

```
flag_field_conflicts(
  data,
  zip = "zip_clean",
  state = "state_clean",
  stated_county = NULL,
  geocoded_county = "location_county"
)
```

Arguments

`data`	A data frame of cleaned/geocoded records.
`zip`	Name of the ZIP column (default `"zip_clean"`). Set to `NULL` to skip the ZIP check.
`state`	Name of the state column (default `"state_clean"`).
`stated_county`	Optional name of a county column supplied in the input. The county check runs only when this is given.
`geocoded_county`	Name of the geocoded county column to compare against (default `"location_county"`).

Details

The ZIP check is deliberately conservative. It compares the ZIP's leading digit against the USPS regional assignment for the stated state, so it only flags a ZIP that is definitively in the wrong region (for example a `"8xxxx"` ZIP recorded in New Jersey). It never flags a same-region near-miss, and it stays silent when the state is unknown or the ZIP is missing, so it does not produce false positives.

The county check compares a stated county column against a geocoded county column (for example `location_county` from `add_county_muni()`), after normalising case and stripping the trailing "County"/"Parish"/"Borough".

Value

`data` with three added columns: `zip_state_conflict` (logical, `NA` when indeterminate), `county_conflict` (logical, `NA` when either county is missing), and `field_conflict` (a `"; "`-joined summary such as `"zip_state"`, `"county"`, or `"zip_state; county"`; `NA` when clean).

Examples

```
df <- data.frame(
  zip_clean = c("07030", "85001"),   # 07 is NJ; 85 is AZ
  state_clean = c("NJ", "NJ")
)
flag_field_conflicts(df)
```

`geocode_address` *Look up a single address and return ranked candidate points*

Description

An interactive, one-shot companion to the batch pipeline: pass a literal address (no data frame) and get back a tibble of candidate matches ranked by geocoder confidence, highest first. Each candidate carries its coordinates, match score, and ArcGIS address type, and - unless turned off - the county and municipality the point falls in. Handy for spot-checking one address, or for letting a reviewer eyeball the plausible locations the cascade would choose from.

Usage

```
geocode_address(
  address,
  city = NULL,
  state = NULL,
  zip = NULL,
  id = NULL,
  min_score = 0,
  max_candidates = 5L,
  geography = TRUE,
  geography_shapes = NULL,
  bbox = NULL,
  quiet = TRUE,
  show_progress = interactive(),
  cache = NULL,
  refresh = FALSE
)
```

Arguments

`address`	Single-line street address as a length-1 character string.
`city`	Optional locality for the address (length-1 character).
`state`	Optional two-letter state abbreviation. When `city` is supplied but `state` is omitted, defaults to `"NJ"` for compatibility.
`zip`	Optional ZIP/postal code. Improves match precision when supplied.
`id`	Optional label echoed back in the `query_id` column.

min_score	Minimum ArcGIS match score (0-100) a returned candidate must reach to be kept. Defaults to 0. This filters ArcGIS results; use city, state, bbox, or zip to change the search context.
max_candidates	Maximum number of candidates to return. Defaults to 5.
geography	If TRUE (default), attach County/Municipality (and the other local-geography fields). When state is not supplied, locatr tries to infer candidate states from ArcGIS matched addresses. Set FALSE for coordinates only.
geography_shapes	Optional sf boundary layer to attach geography from (via add_muni_from_shapes()). When NULL and geography = TRUE, county subdivisions are built from Census TIGER/Line for state (needs tigris and network access).
bbox	Optional region bounding box (see region_bbox()). When given, ArcGIS is asked to prefer that extent and an in_bbox flag is added.
quiet	If TRUE (default), suppress routine messages from geography downloads/joins so the console shows only the returned candidate table.
show_progress	If TRUE, print short progress messages while the lookup runs. Defaults to interactive().
cache	Optional locatr_cache() object. When supplied, the ArcGIS candidate lookup for a given query is served from the cache on repeat calls (and replayable offline) instead of re-hitting the service.
refresh	If TRUE, bypass any cached entry for this query and re-query the service, overwriting the cached result. Defaults to FALSE.

Details

The address text is normalised with the same abbreviation/secondary-unit cleaning used by clean_addresses(), then sent to the free ArcGIS findAddressCandidates service. city, state, and zip are optional for this one-off helper: use them when you want to narrow the search, or pass only address to inspect broad candidate matches. If city is supplied and state is omitted, state defaults to "NJ" for compatibility with the package's first workflow.

Value

A tibble of candidates ordered by descending match_score, with query_id, rank, match_score, match_addr_type, matched_address, latitude, longitude, the cleaned input_address, an optional in_bbox flag, and (when geography = TRUE) County/Municipality and related fields. Zero rows if nothing matched at or above min_score.

See Also

geocode_records() for the batch cascade over a data frame.

Examples

```
if (interactive()) {
# ranked candidates for one address
geocode_address("1600 Pennsylvania Ave NW")
```

```
# only high-confidence matches, coordinates only
geocode_address("1 City Hall Sq", city = "Boston", state = "MA",
                min_score = 90, geography = FALSE)
}
```

`geocode_arcgis` *ArcGIS address fallback pass (Google-like fuzzy matching)*

Description

For rows the Census pass could not place inside the configured region, re-geocodes with a composite geocoder (ArcGIS by default: free, no API key, fuzzy matching close to Google) using the single-line `full_address_clean`. ArcGIS requests are constrained to the region bbox when possible, and results are still guarded against the bounding box so out-of-region false matches are discarded before coordinates are coalesced back into `latitude`/`longitude`.

Usage

```
geocode_arcgis(
  data,
  method = "arcgis",
  bbox = region_bbox("NJ"),
  ...,
  cache = NULL,
  refresh = FALSE
)
```

Arguments

`data`	A data frame from `geocode_census()` (or after `validate_geocodes()`).
`method`	tidygeocoder method for this pass (default `"arcgis"`). `"google"` also works if `GOOGLEGEOCODE_API_KEY` is set.
`bbox`	Bounding box used to reject out-of-region matches; see `region_bbox()`.
`...`	Passed through to `tidygeocoder::geocode()`.
`cache`	Optional `locatr_cache()`. When supplied, the ArcGIS lookup for a given `full_address_clean` (under the same region extent) is served from the cache instead of re-querying.
`refresh`	If `TRUE`, ignore cached entries and re-query, overwriting them. Defaults to `FALSE`.

Details

Formerly `geocode_fallback()`; renamed because this tier is specifically the ArcGIS (composite) address pass.

Value

data with fallback columns fb_latitude, fb_longitude, fb_status, and updated latitude, longitude, geocode_method, geocode_pass, match_status for rows this pass filled.

---

geocode_by_name *Name-based geocode pass (the "paste it in a browser" tier)*

Description

For rows still unplaced after the address-based passes, geocodes by record *name* plus city and state rather than the street line. This can resolve campus/landmark addresses (e.g. a unit inside a hospital) that street-range interpolation cannot, because a composite geocoder recognises the named place.

Usage

```
geocode_by_name(
  data,
  method = "arcgis",
  bbox = region_bbox("NJ"),
  min_score = 90,
  accept_types = c("PointAddress", "Subaddress", "StreetAddress"),
  ...,
  cache = NULL,
  refresh = FALSE
)
```

Arguments

data	A data frame carrying record_id, record_name, city_clean, state_clean, and the geocode audit columns.
method	tidygeocoder method that accepts free-text queries (default "arcgis"; "osm" and "google" also work).
bbox	Bounding box used to reject out-of-region matches; see region_bbox().
min_score	Minimum match score (0-100) for a name hit to pass without review. Hits below this stay reviewable. Default 90.
accept_types	Address types precise enough to pass without review (matched case-insensitively against the geocoder's addr_type). Default the point-address types c("PointAddress", "Subaddress", "StreetAddress").
...	Passed through to tidygeocoder::geocode(). full_results = TRUE is requested automatically so scores are available; pass full_results = FALSE to opt out (which also disables score gating).
cache	Optional locatr_cache(). When supplied, the name lookup for a given query (under the same region extent) is served from the cache, including its cached score and address type, instead of re-querying.
refresh	If TRUE, ignore cached entries and re-query, overwriting them. Defaults to FALSE.

Details

Because name lookups are looser than address matching, each hit is scored using the geocoder's match `score` and address type (when available - ArcGIS returns both via `full_results`, which this pass requests automatically). A hit passes cleanly only when it resolves to a precise point address at or above `min_score`; fuzzier hits (a POI, a locality centroid, or a low score) still have their coordinates filled in for context but are marked `match_status == "matched_low_confidence"` and routed to `needs_manual_review` so a person can confirm them. When the geocoder returns no score/type information (e.g. `method = "osm"`), the pass falls back to the previous rule: any in-region match is accepted.

Filled rows are tagged `geocode_pass == "pass_4_name_lookup"`. The bounding box still rejects out-of-region hits, but cannot catch a wrong same-state match, which is exactly what the score gate is for.

Value

`data` with name-lookup audit columns `nm_latitude`, `nm_longitude`, `nm_score`, `nm_addr_type`, `nm_status`, and updated `latitude`/`longitude`/`geocode_method`/`geocode_pass`/`match_status` for rows the name pass filled. When there is nothing for the tier to geocode, `nm_status` is set to `"not_run"` for audit clarity. Low-confidence fills also set `review_status == "needs_manual_review"`.

`geocode_census` *Primary geocode pass via the US Census batch geocoder*

Description

Geocodes only the rows marked `ready_for_geocoding`, using the *structured* Census engine (street / city / state / ZIP) rather than a single-line string, which matches reliably more often. Rows not ready are returned untouched with empty coordinate columns so the frame stays rectangular.

Usage

```
geocode_census(data, ..., cache = NULL, refresh = FALSE)
```

Arguments

`data`	A data frame from `flag_bad_addresses()`.
`...`	Passed through to `tidygeocoder::geocode()` (e.g. `full_results`).
`cache`	Optional `locatr_cache()`. When supplied, rows whose structured query is already cached are filled from it instead of re-querying Census.
`refresh`	If `TRUE`, ignore cached entries and re-query, overwriting them. Defaults to `FALSE`.

Details

Volatile Census full-result columns (`tiger_line_id`, `id`) are coerced to character to avoid the `bind_rows()` integer/character type clash that the Census service triggers intermittently between batches.

Value

data with latitude, longitude, geocode_method, geocode_pass, match_status, plus Census full-result columns when full_results = TRUE (full-result columns are not stored in the cache, so cache-filled rows omit them).

---

geocode_provenance *Read the provenance manifest from a geocoding run*

Description

geocode_records() attaches a run manifest as an attribute of its output. This returns it: a run id and UTC timestamp, the locatr / tidygeocoder / cache-schema versions, the tiers run, whether a reference table was used, the cache path, per-review_status counts, and cache activity (cache_hits / cache_misses / cache_writes). Read it directly on the geocode_records() result, since later data-frame operations may drop the attribute.

Usage

```
geocode_provenance(data)

## S3 method for class 'locatr_provenance'
print(x, ...)
```

Arguments

data	Output of geocode_records().
x	A locatr_provenance object.
...	Ignored.

Value

A locatr_provenance object (a named list) describing the run.

See Also

geocode_records(), locatr_cache()

geocode_records *Run the full geocoding cascade*

Description

Orchestrates the tiered strategy on an already-cleaned, already-flagged frame (see `clean_addresses()` and `flag_bad_addresses()`): Census structured match, then ArcGIS address fallback, then name lookup, validating against the configured region after each tier so a later tier only retries what is still unplaced.

Usage

```
geocode_records(
  data,
  tiers = c("census", "arcgis", "name"),
  reference = NULL,
  boundary = NULL,
  bbox = region_bbox("NJ"),
  name_min_score = 90,
  name_accept_types = c("PointAddress", "Subaddress", "StreetAddress"),
  verbose = TRUE,
  cache = NULL,
  refresh = FALSE
)
```

Arguments

`data`	A data frame from `flag_bad_addresses()`.
`tiers`	Which tiers to run, in order. Any subset of `c("census", "arcgis", "name")`.
`reference`	Optional trusted key -> coordinates table for Tier 0; see `backfill_from_reference()`. `NULL` skips Tier 0. For non-default column names, call `backfill_from_reference()` yourself before `geocode_records()`.
`boundary`	Optional `sf` boundary for polygon-precise validation; passed to `validate_geocodes()`. `NULL` uses the bounding box.
`bbox`	Bounding box for region guards; see `region_bbox()`.
`name_min_score`	Minimum ArcGIS score for a name lookup to pass without review. Passed to `geocode_by_name()`.
`name_accept_types`	ArcGIS address types precise enough for a name lookup to pass without review. Passed to `geocode_by_name()`.
`verbose`	Whether to print a per-tier match tally.
`cache`	Optional `locatr_cache()` shared across the network tiers (Census, ArcGIS, name lookup), so repeated addresses are served from the cache and a re-run reproduces coordinates without re-querying.
`refresh`	If `TRUE`, ignore cached entries and re-query every tier, overwriting the cache. Defaults to `FALSE`.

Details

Each tier records how it placed a row in `geocode_pass`, so the final frame is self-documenting: `pass_0_reference`, `pass_1_census_structured`, `pass_2_fallback`, or `pass_4_name_lookup`. After the cascade, valid matched rows are marked `review_status == "auto_accepted"`; anything still unmatched lands in `needs_manual_review`, while invalid coordinates are `rejected`.

Supplying `reference` runs an authoritative Tier 0 first (`backfill_from_reference()`): rows whose verified coordinates are already known are placed exactly and skipped by every later tier. Feed prior cycles' completed overrides back in here so manual review accrues over time.

Value

`data` with coordinates and the full audit trail populated.

`geocode_report` *Summarise a geocoding run into a provenance report*

Description

Turns a finished `geocode_records()` frame into an audit report: counts by review status, by placing tier, and by cache status; a `match_confidence` summary; and an auto-generated plain-language *methods paragraph* suitable for a report or a paper. When the run manifest is present (attached by `geocode_records()`), the methods paragraph names the package versions, run date, region guard, and cache activity; without it, the report is built from the audit columns alone.

Usage

```
geocode_report(data, file = NULL, low_confidence_below = 0.5)

## S3 method for class 'locatr_report'
print(x, ...)
```

Arguments

`data`	A finished data frame from `geocode_records()` (ideally still carrying its `locatr_run` manifest). At minimum, `review_status` and `geocode_pass` drive the summary; `match_confidence` and `cache_status` are used when present.
`file`	Optional path. When given, a Markdown version of the report is written there and the report object is returned invisibly.
`low_confidence_below`	Confidence threshold (0-1) used to count low-confidence rows in the summary. Defaults to `0.5`.
`x`	A `locatr_report` object.
`...`	Ignored.

Value

A `locatr_report` object (a named list) with the counts, confidence summary, and `methods` paragraph. Printing it shows a formatted summary.

See Also

`geocode_records()`, `geocode_provenance()`, `add_match_confidence()`

Examples

```
df <- data.frame(
  record_id = c("a", "b", "c"),
  geocode_pass = c("pass_1_census_structured", "pass_2_fallback",
                   "pass_4_name_lookup"),
  review_status = c("auto_accepted", "auto_accepted", "needs_manual_review"),
  match_confidence = c(0.9, 0.72, 0.35)
)
geocode_report(df)
```

---

in_bbox *Is a coordinate inside a bounding box?*

Description

Is a coordinate inside a bounding box?

Usage

```
in_bbox(lat, lon, bbox)
```

Arguments

`lat`	Numeric vector of latitudes.
`lon`	Numeric vector of longitudes.
`bbox`	A named bounding box as returned by `region_bbox()` or supplied by the caller.

Value

A logical vector the same length as `lat`/`lon`. `NA` coordinates return `FALSE`.

Examples

```
in_bbox(40.2, -74.5, region_bbox("NJ"))
in_bbox(40.5, -104.9, region_bbox("NJ")) # a Colorado false-match
```

`locatr_cache` *Create a locatr response cache*

Description

A cache of parsed geocoder results that makes runs reproducible: repeated queries are served locally instead of re-hitting the service, and - because the parsed coordinates are stored - a cached result can be replayed offline, even without the `httr`/`jsonlite` packages that the live call needs. Pass the returned object to `geocode_address()` via its `cache` argument.

Usage

```
locatr_cache(path = NULL, format = c("rds", "parquet"), store_raw = FALSE)

## S3 method for class 'locatr_cache'
print(x, ...)
```

Arguments

`path`	Optional file path for a persistent cache. `NULL` (default) keeps the cache in memory for the session only - no disk writes. When a path is given, the cache is loaded from it if present and flushed on every write.
`format`	On-disk format when `path` is set: `"rds"` (default, no extra dependency) or `"parquet"` (needs `arrow`).
`store_raw`	Reserved for storing raw service responses. `FALSE` by default; raw storage is only ever kept for services whose terms permit it.
`x`	A `locatr_cache` object.
`...`	Ignored.

Details

The cache table is *long*: one row per candidate result (so `geocode_address()`'s ranked set round-trips exactly), plus a single sentinel row (`result_rank = 0`, `status = "no_match"`) for a query that matched nothing, so misses are replayable and never silently re-queried.

The lookup `key` is an implementation detail (a hash). The visible `method`, `endpoint`, `query`, and `params` columns are the audit contract - keys can always be rebuilt from them if a future `rlang` changes its hash.

Value

A `locatr_cache` object (an environment) to pass to geocoding functions.

See Also

`cache_info()`, `cache_clear()`, `geocode_address()`

Examples

```
cache <- locatr_cache()
cache_info(cache)
```

`plot_geocode_review_map` *Plot geocoded records for review*

Description

Creates a small interactive leaflet map, colored by an audit column. The helper is intentionally lightweight: it is for quick review, not for producing a full dashboard.

Usage

```
plot_geocode_review_map(
  data,
  color_by = c("review_status", "geocode_pass", "match_status")
)
```

Arguments

`data`	A locatr output data frame with `latitude` and `longitude`.
`color_by`	Column used to color points. Defaults to `review_status`.

Value

A `leaflet` map.

Examples

```
if (interactive()) {
  plot_geocode_review_map(geocoded)
}
```

`region_bbox` *Region bounding box*

Description

Returns an approximate latitude/longitude bounding box for a US state (or `"DC"`), used as a fast sanity guard on geocoded coordinates. Presets are deliberately a little generous so legitimate edge locations are not rejected; they are coarse guard boxes, not precise boundaries. For a tighter or non-state region, pass your own named vector, or derive one from an `sf` layer with `bbox_from_sf()`.

Usage

```
region_bbox(region = "NJ")
```

Arguments

region	Two-letter US state abbreviation (or "DC"), case-insensitive. Defaults to "NJ".

Value

A named numeric vector with elements lat_min, lat_max, lon_min, lon_max.

Examples

```
region_bbox("NJ")
region_bbox("CA")
```

---

run_locatr_app *Launch the locatr demo Shiny app*

Description

Runs the bundled web app (the same one published as a Hugging Face Space): upload a CSV/Excel/Parquet file, geocode it with the locatr cascade and a session cache, download the geocoded records directly, or optionally attach local geography from Census TIGER/Line/an uploaded shapefile before downloading a crosswalk. The app can add Census policy geographies, flag ZIP/state and county conflicts, remove selected columns before export, and download a provenance/reporting bundle for audit records. The app is for demonstration and light interactive use; production pipelines should call the package functions directly.

Usage

```
run_locatr_app(...)
```

Arguments

...	Passed to shiny::runApp() (e.g. port, host, launch.browser).

Details

The app depends on packages that are only listed under Suggests, so they are not installed automatically. If any are missing, this function reports their names and stops.

Value

Called for its side effect of starting the app; does not return.

Examples

```
if (interactive()) {
run_locatr_app()
}
```

`run_locatr_review_app` *Launch the locatr map-based review app*

Description

A standalone Shiny app that takes records from upload through geocoding to map-based review. Upload a file, map the ID and address columns, then either geocode the addresses (clean, flag, cascade, and attach county/municipality) or use existing coordinates. The flagged records appear on a map to accept, reject, or relocate; the app builds a completed override table with `build_review_overrides()` that you download as CSV and feed to `apply_manual_overrides()`. Geocoding calls external services and needs network access.

Usage

```
run_locatr_review_app(...)
```

Arguments

`...`	Passed to `shiny::runApp()` (e.g. `port`, `host`, `launch.browser`).

Details

The app depends on packages listed only under `Suggests`, so they are not installed automatically. If any are missing, this function reports their names and stops.

Value

Called for its side effect of starting the app; does not return.

See Also

`build_review_overrides()`, `apply_manual_overrides()`

Examples

```
if (interactive()) {
run_locatr_review_app()
}
```

suggest_geography_level *Suggest a Census geography level for a state*

Description

Gives a practical starting point for `build_local_geography()` and `add_county_muni()`. It is a recommendation, not a legal definition of local government. For production municipal joins, an official state/local boundary layer is still the strongest source.

Usage

```
suggest_geography_level(state)
```

Arguments

`state`	Two-letter state abbreviation.

Value

A one-row tibble with the recommended Census geography and note.

Examples

```
suggest_geography_level("NJ")
suggest_geography_level("CA")
```

summarise_geocoding *Summarise geocoding quality*

Description

Counts the main outcomes in a locatr result so you can quickly judge whether a run is ready for review, export, or threshold tuning.

Usage

```
summarise_geocoding(data)
```

Arguments

`data`	A locatr output data frame.

Value

A one-row tibble with counts and rates.

Examples

```
x <- tibble::tibble(
  latitude = c(40, NA),
  longitude = c(-75, NA),
  match_status = c("matched", "unmatched"),
  review_status = c("auto_accepted", "needs_manual_review"),
  geocode_pass = c("pass_1_census_structured", NA_character_)
)
summarise_geocoding(x)
```

---

validate_geocodes *Validate geocoded coordinates against a region*

Description

Rejects suspicious coordinates before they reach a dashboard. By default this is a fast bounding-box check; supply `boundary` (an `sf` polygon of the service area) for a precise point-in-polygon test.

Usage

```
validate_geocodes(data, boundary = NULL, bbox = region_bbox("NJ"))
```

Arguments

`data`	A geocoded data frame with `latitude`/`longitude`.
`boundary`	Optional `sf` polygon. When given, validation uses point-in-polygon instead of the bounding box.
`bbox`	Bounding box for the fast path; see `region_bbox()`.

Value

`data` with `validation_status` and an updated `review_status` (anything failing validation or unmatched becomes `needs_manual_review`).

---

write_geocode_review *Export only the records that still need a human*

Description

Writes a tidy review CSV of rows whose `review_status` is `"needs_manual_review"`, with blank `manual_*` columns for a reviewer to fill in. Feed the completed file back through `apply_manual_overrides()`.

Usage

```
write_geocode_review(data, path)
```

Arguments

`data`	A data frame carrying the audit columns.
`path`	Output CSV path.

Value

Invisibly, the review tibble that was written.

Index