# Package: PNADCperiods (via r-universe)

June 3, 2026

**Title** Identify Reference Periods in Brazil's PNADC Survey Data

**Version** 0.1.2

**Date** 2026-04-27

**Description** Identifies reference periods (months, fortnights, and weeks) in Brazil's quarterly PNADC (Pesquisa Nacional por Amostra de Domicilios Continua) survey data and computes calibrated weights for sub-quarterly analysis. The core algorithm uses IBGE (Instituto Brasileiro de Geografia e Estatistica) 'Parada Tecnica' (technical break) rules combined with respondent birthdates to determine which temporal period each survey observation refers to. Period identification follows a nested hierarchy enforced by construction: fortnights require months, weeks require fortnights. Achieves approximately 97% monthly determination rate with the full series (2012-2025). Strict fortnight and week rates are approximately 9% and 3% respectively, as they cannot leverage cross-quarter panel aggregation. Experimental strategies (probabilistic assignment and UPA (Primary Sampling Unit) aggregation) further improve these determination rates. The package provides adaptive hierarchical weight calibration (4/2/1 cell levels for month/fortnight/week) with period-specific smoothing to produce survey weights calibrated to SIDRA (Sistema IBGE de Recuperacao Automatica) population totals. Also includes a SIDRA mensalization module that converts 86+ official rolling quarter series from the IBGE SIDRA API (Application Programming Interface) into exact monthly estimates, without requiring access to microdata. Hecksher and Barbosa (2026) <https://osf.io/preprints/socarxiv/fra5u_v1>.

**License** MIT + file LICENSE

**Language** en-US

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Depends** R (>= 4.1.0)

**Imports** data.table (>= 1.14.0), checkmate (>= 2.0.0), sidrar (>= 0.2.9), lubridate (>= 1.9.4)

**Suggests** dplyr, fst, haven, testthat (>= 3.0.0), knitr, rmarkdown, pkgdown, ggplot2, scales

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**URL** https://antrologos.github.io/PNADCperiods/, https://github.com/antrologos/PNADCperiods

**BugReports** https://github.com/antrologos/PNADCperiods/issues

**Config/pak/sysreqs** libicu-dev libxml2-dev libssl-dev

**Repository** https://antrologos.r-universe.dev

**Date/Publication** 2026-05-04 20:35:47 UTC

**RemoteUrl** https://github.com/antrologos/pnadcperiods

**RemoteRef** HEAD

**RemoteSha** 34d35ba3dfad3d96ae78acffe356b69c8b583bcd

# Contents

## clear_sidra_cache *Clear All SIDRA Caches*

### Description

Clears all cached SIDRA data (both rolling quarter series and population). Use this if you need to force a fresh download, for example after IBGE updates their data.

### Usage

```
clear_sidra_cache()
```

### Value

Invisibly returns TRUE if any cache was cleared, FALSE if all empty.

### Examples

```
clear_sidra_cache()
```

## compute_series_starting_points *Compute Starting Points from Microdata*

### Description

For advanced users who want to compute custom starting points using their own calibrated microdata estimates.

### Usage

```
compute_series_starting_points(
  monthly_estimates,
  rolling_quarters,
  calibration_start = NULL,
  calibration_end = NULL,
  scale_factor = 1000,
  use_series_specific_periods = TRUE,
  verbose = TRUE
)
```

## Arguments

`monthly_estimates`
: data.table with columns:
  - `anomesexato`: YYYYMM exact month
  - `z_*`: Monthly estimates from calibrated microdata (one column per series)

`rolling_quarters`
: data.table from `fetch_sidra_rolling_quarters`

`calibration_start`
: Integer. Start of calibration period (YYYYMM). Default NULL uses .PNADC_DATES$DEFAULT_CAL (201301). Note: CNPJ series automatically use CNPJ_CALIB_START (201601) regardless.

`calibration_end`
: Integer. End of calibration period (YYYYMM). Default NULL uses .PNADC_DATES$DEFAULT_CALI (201912).

`scale_factor`
: Numeric. Scale factor for z_ values (usually 1000). Default 1000.

`use_series_specific_periods`
: Logical. If TRUE (default), use series-specific calibration periods for CNPJ series (201601-201912) and cumsum starting dates (201510). Set to FALSE to use uniform calibration for all series.

`verbose`
: Logical. Print progress? Default TRUE.

## Details

The starting points (y0) are computed by:

1. Calculating cumulative variations from SIDRA rolling quarters
2. Computing backprojection: e0 = z / scale_factor - cum
3. Averaging e0 by mesnotrim over the calibration period

## Value

data.table with columns:

**series_name** Character. Series name

**mesnotrim** Integer. Month position (1, 2, or 3)

**y0** Numeric. Starting point value

## Series-Specific Handling

When `use_series_specific_periods = TRUE`, the following series receive special handling for series-specific data availability:

**CNPJ series** empregadorcomcnpj, empregadorsemcnpj, contapropriacomcnpj, contapropriasemcnpj use calibration period 201601-201912 and cumsum starts from 201510 (when V4019 became available)

## Examples

```
## Not run:
rq <- fetch_sidra_rolling_quarters()
z_agg <- compute_z_aggregates(calibrated_data)
y0 <- compute_series_starting_points(z_agg, rq)
monthly <- mensalize_sidra_series(rq, starting_points = y0)

## End(Not run)
```

---

`compute_starting_points_from_microdata`
*Compute Starting Points from Raw PNADC Microdata*

---

## Description

Complete workflow to compute y0 starting points from raw PNADC microdata. This is a convenience wrapper that combines period identification, weight calibration, z_ aggregation, and starting point computation.

## Usage

```
compute_starting_points_from_microdata(
  data,
  calibration_start = NULL,
  calibration_end = NULL,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | Stacked PNADC microdata (multiple quarters). Must contain variables for period identification (see `pnadc_identify_periods`). |
| `calibration_start` | Integer. Start of calibration period (YYYYMM). Default NULL uses .PNADC_DATES$DEFAULT_CAL (201301). |
| `calibration_end` | Integer. End of calibration period (YYYYMM). Default NULL uses .PNADC_DATES$DEFAULT_CALI (201912). |
| `verbose` | Print progress messages. |

## Details

This function performs the complete workflow:

1. Build crosswalk via `pnadc_identify_periods()`
2. Calibrate weights via `pnadc_apply_periods()`

3. Compute z_ aggregates via `compute_z_aggregates()`
4. Fetch SIDRA rolling quarters
5. Compute starting points via `compute_series_starting_points()`

## Value

data.table with columns:

**series_name** Character. Series name

**mesnotrim** Integer. Month position (1, 2, or 3)

**y0** Numeric. Starting point value

## Weight Calibration

All months are scaled uniformly to SIDRA monthly population totals.

## See Also

`pnadc_apply_periods` for the weight calibration step `compute_z_aggregates` for the z_ aggregation step `compute_series_starting_points` for the y0 computation `pnadc_identify_periods` for period identification

## Examples

```
## Not run:
stacked <- fst::read_fst("pnadc_stacked.fst", as.data.table = TRUE)

y0 <- compute_starting_points_from_microdata(stacked)

bundled <- pnadc_series_starting_points
comparison <- merge(y0, bundled, by = c("series_name", "mesnotrim"))

## End(Not run)
```

---

# `compute_z_aggregates` *Compute z_ Aggregates from Monthly Microdata*

---

## Description

Computes monthly z_ aggregates from PNADC microdata using calibrated monthly weights, with options for different population scaling approaches.

## Usage

```
compute_z_aggregates(calibrated_data, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| calibrated_data | PNADC microdata output from pnadc_apply_periods() with calibrate = TRUE. Must include weight_monthly, ref_month_yyyymm, and ref_month_in_quarter. |
| verbose | Print progress messages. |

## Details

This function creates z_ indicator variables and aggregates them using the calibrated weight_monthly from pnadc_apply_periods().

The pnadc_apply_periods() function implements the calibration methodology as follows: All months are scaled uniformly to SIDRA monthly population totals.

This function simply aggregates the indicators using the already-calibrated weights.

## Value

data.table with columns:

**anomesexato** Integer YYYYMM month

**z_*** Numeric weighted aggregates for each series (one column per series)

## See Also

pnadc_apply_periods for the calibration step compute_series_starting_points for the y0 computation

## Examples

```
## Not run:
crosswalk <- pnadc_identify_periods(stacked_data)

calibrated <- pnadc_apply_periods(stacked_data, crosswalk,
                                  weight_var = "V1028",
                                  anchor = "quarter",
                                  calibration_unit = "month")

z_agg <- compute_z_aggregates(calibrated)

rq <- fetch_sidra_rolling_quarters()
y0 <- compute_series_starting_points(z_agg, rq)

## End(Not run)
```

`fetch_monthly_population` *Fetch Monthly Population from SIDRA*

## Description

Downloads population estimates from IBGE SIDRA API (table 6022) and transforms from moving-quarter to exact monthly values.

## Usage

```
fetch_monthly_population(
  start_yyyymm = NULL,
  end_yyyymm = NULL,
  verbose = TRUE,
  use_cache = FALSE,
  cache_max_age_hours = 24
)
```

## Arguments

| | |
|---|---|
| `start_yyyymm` | Integer. First month to include (YYYYMM format). If NULL, returns all available months. |
| `end_yyyymm` | Integer. Last month to include (YYYYMM format). If NULL, returns all available months. |
| `verbose` | Logical. Print progress messages? Default TRUE. |
| `use_cache` | Logical. If TRUE, uses cached data if available and not expired. Default FALSE (always fetch fresh data for consistency). Set to TRUE for faster repeated calls during development. |
| `cache_max_age_hours` | Numeric. Maximum cache age in hours before automatic expiration when use_cache=TRUE. Default 24 hours. |

## Details

SIDRA table 6022 provides moving-quarter population estimates. Each value represents the 3-month average centered on the middle month. For example, the value for code 201203 (quarter ending March 2012) represents the population for February 2012.

This function:

1. Fetches raw moving-quarter data from SIDRA
2. Transforms to exact monthly values by aligning with middle months
3. Extrapolates boundary months (first and last) using quadratic regression

The extrapolation uses quadratic regression on population differences to estimate the first month (Jan 2012) and the most recent month.

## Value

A data.table with columns:

- `ref_month_yyyymm`: Integer in YYYYMM format
- `m_populacao`: Monthly population in thousands

Returns `NULL` invisibly with an informative message if the SIDRA API is unreachable (per CRAN policy on Internet resources).

## Dependencies

This function requires the `sidrar` package for API access. Install with: `install.packages("sidrar")`

## See Also

`pnadc_apply_periods` which uses this function when `calibrate = TRUE`

## Examples

```
pop <- fetch_monthly_population()

pop <- fetch_monthly_population(201301, 201912)
```

---

# fetch_sidra_rolling_quarters

*Fetch Rolling Quarter Series from SIDRA*

---

## Description

Downloads PNADC labor market indicators from IBGE's SIDRA API. These series are published as rolling quarterly averages (trimestre movel), with 12 observations per year.

## Usage

```
fetch_sidra_rolling_quarters(
  series = "all",
  theme = NULL,
  theme_category = NULL,
  subcategory = NULL,
  exclude_derived = FALSE,
  use_cache = FALSE,
  verbose = TRUE,
  retry_failed = TRUE,
  max_retries = 3
)
```

## Arguments

| | |
|---|---|
| `series` | Character vector of series names to fetch, or "all" (default) for all available series. Use `get_sidra_series_metadata()$series_name` to see available names. |
| `theme` | Character vector of themes to filter by. Valid options: "labor_market", "earnings", "demographics", "social_protection", "prices". Use NULL for no filter. |
| `theme_category` | Character vector of theme categories to filter by. Use NULL for no filter. |
| `subcategory` | Character vector of subcategories to filter by. Use NULL for no filter. |
| `exclude_derived` | Logical. If TRUE, exclude series marked as derived (is_derived = TRUE in metadata). Default FALSE for backward compatibility. Derived series (rates) are computed from other series during mensalization, so excluding them saves API calls when fetching for mensalization. |
| `use_cache` | Logical. Use cached data if available? Default FALSE. When TRUE, shows the date when data was cached (may be outdated). Use `clear_sidra_cache` to force fresh download. |
| `verbose` | Logical. Print progress messages? Default TRUE. |
| `retry_failed` | Logical. Retry failed series downloads? Default TRUE. |
| `max_retries` | Integer. Maximum retry attempts per series. Default 3. |

## Details

Rolling quarters are labeled by their ending month:

- 201201 = Nov 2011 - Jan 2012 (mesnotrim = 1)
- 201202 = Dec 2011 - Feb 2012 (mesnotrim = 2)
- 201203 = Jan - Mar 2012 (mesnotrim = 3)
- 201204 = Feb - Apr 2012 (mesnotrim = 1)
- etc.

The `mesnotrim` column indicates the month's position within its rolling quarter, which is essential for the mensalization algorithm.

## Value

A data.table with columns:

**anomesfinaltrimmovel** Integer. YYYYMM of rolling quarter end month

**mesnotrim** Integer. Month position in quarter (1, 2, or 3)

**<series_name>** Numeric. One column per requested series

## Rate Limiting

SIDRA API may have rate limits. The function includes automatic retry logic with exponential backoff for failed requests.

### Internet Resource Behaviour

Per CRAN policy, this function fails gracefully when the SIDRA API is unreachable: it emits an informative `message()` (no warning, no error) and returns `NULL` invisibly when no series could be fetched. Partial failures (some series succeeded) are also reported via `message()`, and the result includes only the successful series.

### See Also

`get_sidra_series_metadata` for available series names and metadata `mensalize_sidra_series` to convert to exact months

### Examples

```
rq <- fetch_sidra_rolling_quarters(
  series = c("taxadesocup", "popocup", "popdesocup")
)
head(rq)



rq_labor <- fetch_sidra_rolling_quarters(theme = "labor_market")
```

---

## get_sidra_series_metadata

*Get SIDRA Series Metadata*

---

### Description

Returns a data.table with metadata for all PNADC rolling quarter series available from IBGE's SIDRA API.

### Usage

```
get_sidra_series_metadata(
  series = "all",
  theme = NULL,
  theme_category = NULL,
  subcategory = NULL,
  lang = "pt"
)
```

## Arguments

| | |
|---|---|
| series | Character vector of series names to retrieve, or "all" (default) for all series. |
| theme | Character vector of themes to filter by. Valid themes: "labor_market", "earnings", "demographics", "social_protection", "prices". Use NULL (default) for no filtering. |
| theme_category | Character vector of theme categories to filter by. Use NULL (default) for no filtering. |
| subcategory | Character vector of subcategories to filter by. Use NULL (default) for no filtering. |
| lang | Character. Language for descriptions: "pt" (Portuguese, default) or "en" (English). When "en", the `description` column contains English text. |

## Value

A data.table with columns:

**series_name** Character. Internal name used in the package

**api_path** Character. SIDRA API path for get_sidra()

**table_id** Integer. SIDRA table number

**variable_id** Integer. SIDRA variable code

**theme** Character. Top-level theme

**theme_category** Character. Middle-level category within theme

**subcategory** Character. Optional subcategory for filtering

**description_pt** Character. Portuguese description

**description_en** Character. English description

**description** Character. Description in the requested language

**unit** Character. Unit of measurement

**unit_label_pt** Character. Unit label in Portuguese

**unit_label_en** Character. Unit label in English

**is_derived** Logical. TRUE if computed from other series

**requires_deflation** Logical. TRUE if needs IPCA deflation

## Examples

```
meta <- get_sidra_series_metadata()
head(meta)

labor <- get_sidra_series_metadata(theme = "labor_market")

unemp <- get_sidra_series_metadata(theme = "labor_market",
                                   theme_category = "unemployment")

meta_en <- get_sidra_series_metadata(series = c("taxadesocup", "popocup"),
                                     lang = "en")
```

`mensalize_sidra_series` *Convert Rolling Quarters to Exact Monthly Series*

## Description

Transforms SIDRA rolling quarterly averages into exact monthly values using the mathematical relationship between consecutive rolling quarters.

## Usage

```
mensalize_sidra_series(
  rolling_quarters,
  starting_points = NULL,
  series = "all",
  compute_derived = TRUE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `rolling_quarters` | data.table from `fetch_sidra_rolling_quarters` containing rolling quarter series with columns `anomesfinaltrimmovel`, `mesnotrim`, and series value columns. |
| `starting_points` | Optional data.table with precomputed starting points (y0 values). If NULL (default), uses bundled `pnadc_series_starting_points`. See Details for format. |
| `series` | Character vector of series names to mensalize, or "all" (default) for all series in the input data (except price indices). |
| `compute_derived` | Logical. Compute derived series (rates, aggregates)? Default TRUE. |
| `verbose` | Logical. Print progress messages? Default TRUE. |

## Details

The algorithm exploits the mathematical property of rolling quarterly averages:

$$RQ_t - RQ_{t-1} = (Month_t - Month_{t-3})/3$$

This means exact 3-month variations can be extracted from consecutive rolling quarters. By accumulating these variations separately for each month-position (1, 2, or 3), we build cumulative variation series. The only unknown is the starting level for Jan, Feb, and Mar 2012.

**Starting points** are estimated by:

1. Computing monthly estimates from calibrated microdata (z_ variables)
2. Calculating cumulative variations from SIDRA (cum_ variables)

3. Backprojecting: e0 = z_ - cum_ over calibration period (2013-2019)
4. Averaging e0 by month position to get y0_ for each position

**Final adjustment** ensures the average of 3 consecutive mensalized values equals the original rolling quarter value.

## Value

A data.table with columns:

**anomesexato** Integer. YYYYMM exact month

**m_*** Numeric. Mensalized value for each series (one column per series)

## Starting Points Format

If providing custom starting points, the data.table must have columns:

- `series_name`: Character. Series name matching rolling_quarters columns
- `mesnotrim`: Integer (1, 2, or 3). Month position in quarter
- `y0`: Numeric. Starting point value

## Mathematical Foundation

The mensalization algorithm proceeds in steps:

1. Calculate d3 = 3 * (RQ_t - RQ_t-1)
2. Separate d3 by month position: d3m1, d3m2, d3m3
3. Cumulate separately: cum1, cum2, cum3
4. Apply starting points: y = y0 + cum
5. Final adjustment for rolling quarter consistency

## See Also

`fetch_sidra_rolling_quarters` to obtain input data `compute_series_starting_points` for custom calibration

## Examples

```
rq <- fetch_sidra_rolling_quarters(
  series = c("taxadesocup", "popocup", "popdesocup")
)
monthly <- mensalize_sidra_series(rq)
head(monthly)
```

pnadc_apply_periods *Apply Reference Period Crosswalk to PNADC Data*

## Description

This function takes a crosswalk from pnadc_identify_periods and applies it to any PNADC dataset (quarterly or annual). It can optionally calibrate the survey weights to match external population totals at the chosen temporal granularity (month, fortnight, or week).

## Usage

```
pnadc_apply_periods(
  data,
  crosswalk,
  weight_var,
  anchor,
  calibrate = TRUE,
  calibration_unit = c("month", "fortnight", "week"),
  calibration_min_cell_size = 1,
  target_totals = NULL,
  smooth = FALSE,
  keep_all = TRUE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| data | A data.frame or data.table with PNADC microdata. Must contain join keys Ano, Trimestre, UPA, V1008, and V1014 to merge with the crosswalk. |
| crosswalk | A data.table crosswalk from pnadc_identify_periods. |
| weight_var | Character. Name of the survey weight column. Must be specified:<br>• "V1028" for quarterly PNADC data<br>• "V1032" for annual PNADC data (visit-specific or annual releases organized by quarters) |
| anchor | Character. How to anchor the weight redistribution. Must be specified:<br>• "quarter" for quarterly data or annual releases organized by quarters (preserves quarterly totals)<br>• "year" for annual visit-specific data (preserves yearly totals) |
| calibrate | Logical. If TRUE (default), calibrate weights to external population totals. If FALSE, only merge the crosswalk without calibration. |
| calibration_unit | Character. Temporal unit for weight calibration. One of "month" (default), "fortnight", or "week". |

`calibration_min_cell_size`
: Integer. Minimum sample size required in a cell for it to be used in hierarchical raking. Cells smaller than this threshold are collapsed to coarser levels. Default: 1 (use all cells).

`target_totals`
: Optional data.table with population targets. If NULL (default), fetches monthly population from SIDRA and derives targets for fortnight/week. Each time period (month, fortnight, or week) is calibrated to the FULL Brazilian population from SIDRA.

  If providing custom targets, the population column (`m_populacao` for months, `f_populacao` for fortnights, `w_populacao` for weeks) must be in **thousands**. The function multiplies by 1000 internally.

`smooth`
: Logical. If TRUE, smooth calibrated weights to remove quarterly artifacts. Smoothing is adapted per time period: monthly (3-period window), fortnight (7-period window), weekly (no smoothing). Default: FALSE.

`keep_all`
: Logical. If TRUE (default), keep all observations including those with undetermined reference periods. If FALSE, drop undetermined rows.

`verbose`
: Logical. If TRUE (default), print progress messages.

## Details

Merges a reference period crosswalk with PNADC microdata and optionally calibrates survey weights for sub-quarterly analysis.

### Weight Calibration:

When `calibrate = TRUE`, the function performs hierarchical rake weighting:

1. Groups observations by nested demographic/geographic cells
2. Iteratively adjusts weights so sub-period totals match anchor-period totals
3. Calibrates final weights against external population totals (FULL Brazilian population)
4. Optionally smooths weights to remove quarterly artifacts

### Population Targets:

All time periods (months, fortnights, and weeks) are calibrated to the FULL Brazilian population from SIDRA. This means:

- Monthly weights sum to the Brazilian population for that month
- Fortnight weights sum to the Brazilian population for the containing month
- Weekly weights sum to the Brazilian population for the containing month

### Hierarchical Raking Levels:

The number of hierarchical cell levels is automatically adjusted based on the calibration unit to avoid sparse cell issues:

- `"month"`: 4 levels (age, region, state, post-stratum) - full hierarchy
- `"fortnight"`: 2 levels (age, region) - simplified for lower sample size
- `"week"`: 1 level (age groups only) - minimal hierarchy for sparse data

### Anchor Period:

The `anchor` parameter determines how weights are redistributed:

- `"quarter"`: Quarterly totals are preserved and redistributed to months/fortnights/weeks
- `"year"`: Yearly totals are preserved and redistributed to months/fortnights/weeks

Use `anchor = "quarter"` with quarterly V1028 weights, and `anchor = "year"` with annual V1032 weights.

## Value

A data.table with the input data plus crosswalk columns:

**ref_month_in_quarter, ref_month_in_year** Month position (1-3 in quarter, 1-12 in year)

**ref_fortnight_in_month, ref_fortnight_in_quarter** Fortnight position (1-2 in month, 1-6 in quarter)

**ref_week_in_month, ref_week_in_quarter** Week position (1-4 in month, 1-12 in quarter)

**ref_month_yyyymm, ref_fortnight_yyyyff, ref_week_yyyyww** Integer period codes

**determined_month, determined_fortnight, determined_week** Logical determination flags

**weight_monthly, weight_fortnight, or weight_weekly** Calibrated weights (if calibrate=TRUE)

## See Also

`pnadc_identify_periods` to build the crosswalk

## Examples

```
## Not run:
crosswalk <- pnadc_identify_periods(pnadc_stacked)

result <- pnadc_apply_periods(
  pnadc_2023,
  crosswalk,
  weight_var = "V1028",
  anchor = "quarter"
)

result <- pnadc_apply_periods(
  pnadc_annual,
  crosswalk,
  weight_var = "V1032",
  anchor = "year"
)

result <- pnadc_apply_periods(
  pnadc_2023,
  crosswalk,
  weight_var = "V1028",
  anchor = "quarter",
  calibration_unit = "week"
)

result <- pnadc_apply_periods(
  pnadc_2023,
```

```
  crosswalk,
  weight_var = "V1028",
  anchor = "quarter",
  calibrate = FALSE
)

## End(Not run)
```

---

`pnadc_experimental_periods` *Experimental Period Identification Strategies*

---

## Description

Three experimental strategies are available, all properly nested by period:

- **probabilistic**: For narrow ranges (2 possible periods), classifies based on where most of the date interval falls. Assigns only when confidence exceeds threshold.
- **upa_aggregation**: Extends strictly identified periods to other observations in the same UPA-V1014 within the quarter, if a sufficient proportion already have strict identification.
- **both**: Sequentially applies probabilistic strategy first, then UPA aggregation on top. Guarantees identification rate >= max of individual strategies.

## Usage

```
pnadc_experimental_periods(
  crosswalk,
  strategy = c("probabilistic", "upa_aggregation", "both"),
  confidence_threshold = 0.9,
  upa_proportion_threshold = 0.5,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `crosswalk` | A crosswalk data.table from `pnadc_identify_periods()` |
| `strategy` | Character specifying which strategy to apply. Options: "probabilistic", "upa_aggregation", "both" |
| `confidence_threshold` | Numeric (0-1). Minimum confidence required to assign a probabilistic period. Used by probabilistic and combined strategies. Default 0.9. |
| `upa_proportion_threshold` | Numeric (0-1). Minimum proportion of UPA observations (within quarter) that must have strict identification with consensus for extending to unidentified observations. Default 0.5. |
| `verbose` | Logical. If TRUE, print progress information. |

## Details

Provides experimental strategies for improving period identification rates beyond the standard deterministic algorithm. All strategies respect the nested identification hierarchy: weeks require fortnights, fortnights require months.

**Nesting Enforcement:**

All strategies enforce proper nesting:

- Fortnights can only be assigned if month is identified (strictly OR experimentally)
- Weeks can only be assigned if fortnight is identified (strictly OR experimentally)

**Probabilistic Strategy:**

For each period type (processed in order: months, then fortnights, then weeks):

1. Check that the required parent period is identified
2. If bounds are narrowed to exactly 2 sequential periods, calculate which period contains most of the date interval
3. Calculate confidence based on the proportion of interval in the likely period (0-1)
4. Only assign if confidence >= `confidence_threshold`

For months: aggregates at UPA-V1014 level across all quarters (like strict algorithm) For fortnights and weeks: works at household level within quarter

**UPA Aggregation Strategy:**

Extends strictly identified periods based on consensus within geographic groups:

- **Months**: Uses UPA level within quarter
- **Fortnights/Weeks**: Uses UPA level within quarter (all households in same UPA are interviewed in same fortnight/week within a quarter)

1. Calculate proportion of observations with strictly identified period
2. If proportion >= `upa_proportion_threshold` AND consensus exists, extend
3. Apply in nested order: months first, then fortnights, then weeks

**Combined Strategy ("both"):**

Sequentially applies both strategies to maximize identification:

1. First, apply the probabilistic strategy (captures observations with narrow date ranges and high confidence)
2. Then, apply UPA aggregation (extends based on strict consensus within UPA/UPA-V1014 groups)

This guarantees that "both" identifies at least as many observations as either individual strategy alone. The strategies operate independently (UPA aggregation considers only strict identifications), so the result is the union of both strategies.

**Integration with Weight Calibration:**

The output can be passed directly to `pnadc_apply_periods()` for weight calibration. The derived columns combine strict and experimental assignments, with strict taking priority. Use the `probabilistic_assignment` flag to filter if you only want strict determinations.

## Value

A modified crosswalk with additional columns. Output is directly compatible with `pnadc_apply_periods()`:

- `ref_month_in_quarter`, `ref_month_in_year`, `ref_month_yyyymm`: Month position (combined strict + experimental, strict takes priority)
- `ref_fortnight_in_month`, `ref_fortnight_in_quarter`, `ref_fortnight_yyyyff`: Fortnight position (combined strict + experimental)
- `ref_week_in_month`, `ref_week_in_quarter`, `ref_week_yyyyww`: Week position (combined strict + experimental)
- `determined_month`, `determined_fortnight`, `determined_week`: TRUE if period is assigned (strictly or experimentally)
- `determined_probable_month`, `determined_probable_fortnight`, `determined_probable_week`: TRUE if period was assigned by probabilistic strategy
- `probabilistic_assignment`: TRUE if any period was assigned experimentally (vs strictly deterministic)
- `week_1_start`, `week_1_end`, ..., `week_4_start`, `week_4_end`: IBGE week boundaries for the assigned month

## Note

These strategies produce "experimental" assignments, not strict determinations. The standard `pnadc_identify_periods()` function should be used for rigorous analysis. Experimental outputs are useful for:

- Sensitivity analysis
- Robustness checks
- Research into identification algorithm improvements

## See Also

`pnadc_identify_periods` to build the crosswalk that this function modifies. `pnadc_apply_periods` to apply period crosswalk and calibrate weights.

## Examples

```
## Not run:
crosswalk <- pnadc_identify_periods(pnadc_data)

crosswalk_exp <- pnadc_experimental_periods(
  crosswalk,
  strategy = "probabilistic",
  confidence_threshold = 0.9
)

crosswalk_exp[, .(
  strict = sum(!is.na(ref_month_in_quarter) & !probabilistic_assignment),
  experimental = sum(probabilistic_assignment, na.rm = TRUE),
  total = sum(determined_month)
)]
```

```
result <- pnadc_apply_periods(pnadc_data, crosswalk_exp,
                              weight_var = "V1028", anchor = "quarter")

strict_only <- crosswalk_exp[
  probabilistic_assignment == FALSE | is.na(probabilistic_assignment)
]

## End(Not run)
```

---

`pnadc_identify_periods` *Identify Reference Periods in PNADC Data*

---

## Description

PNADC is a quarterly survey, but each interview actually refers to a specific temporal period within the quarter. This function identifies which month, fortnight (quinzena), and week each observation belongs to, enabling sub-quarterly time series analysis.

The algorithm uses a **nested identification approach**:

- **Phase 1**: Identify MONTHS for all observations using:
  - IBGE's reference week timing rules (first reference week – ending in a Saturday – with sufficient days)
  - Respondent birthdates to constrain possible interview dates
  - UPA-panel level aggregation across ALL quarters (panel design)
  - Dynamic exception detection (identifies quarters needing relaxed rules)
- **Phase 2**: Identify FORTNIGHTS for month-determined observations:
  - Search space constrained to 2 fortnights within determined month
  - Household-level aggregation within each quarter
- **Phase 3**: Identify WEEKS for fortnight-determined observations:
  - Search space constrained to ~2 weeks within determined fortnight
  - Household-level aggregation within each quarter

## Usage

```
pnadc_identify_periods(data, verbose = TRUE, store_date_bounds = FALSE)
```

## Arguments

`data` A data.frame or data.table with PNADC microdata. Required columns:

- `Ano`: Survey year
- `Trimestre`: Quarter (1-4)
- `UPA`: Primary Sampling Unit

- `V1008`: Household id/sequence within UPA
- `V1014`: Panel identifier
- `V2008`: Birth day (1-31)
- `V20081`: Birth month (1-12)
- `V20082`: Birth year
- `V2009`: Age

Optional but recommended:

- `V2003`: Person sequence within household

`verbose` Logical. If TRUE (default), display progress information.

`store_date_bounds`
Logical. If TRUE, stores date bounds and exception flags in the crosswalk for optimization when calling `pnadc_experimental_periods()`. This enables 10-20x speedup for the probabilistic strategy by avoiding redundant computation. Default FALSE.

## Details

Builds a crosswalk containing reference periods (month, fortnight, and week) for PNADC survey data based on IBGE's interview timing rules.

**Temporal Granularity:**

The crosswalk contains three levels of temporal granularity:

- **Month**: 3 per quarter, ~97\
- **Fortnight (quinzena)**: 6 per quarter, ~9\
- **Week**: 12 per quarter, ~3\

**Cross-Quarter Aggregation (Important!):**

**For optimal month determination rates, input data should be stacked across multiple quarters** (ideally 4+ years). The algorithm leverages PNADC's rotating panel design where the same UPA-V1014 is interviewed in the same relative position across quarterly visits.

**Fortnight Definition:**

Fortnights are numbered 1-6 per quarter (2 per month), based on the IBGE reference week calendar (not calendar days). Each IBGE "month" consists of exactly 4 reference weeks (28 days), starting on a Sunday:

- Fortnight 1 in month: IBGE weeks 1-2 (days 1-14 of the IBGE month)
- Fortnight 2 in month: IBGE weeks 3-4 (days 15-28 of the IBGE month)

## Value

A data.table crosswalk with columns:

**Ano, Trimestre, UPA, V1008, V1014** Join keys (year, quarter, UPA, household, panel)

**ref_month_in_quarter** Integer. Month position in quarter (1, 2, 3) or NA

**ref_month_in_year** Integer. Month position in year (1-12) or NA

**ref_fortnight_in_month** Integer. Fortnight position in month (1 or 2) or NA

**ref_fortnight_in_quarter** Integer. Fortnight position in quarter (1-6) or NA

**ref_week_in_month** Integer. Week position in month (1-4) or NA

**ref_week_in_quarter** Integer. Week position in quarter (1-12) or NA

**date_min** Date. Lower bound of the interview reference date for the individual. Only returned if store_date_bounds = TRUE

**date_max** Date. Upper bound of the interview reference date for the individual. Only returned if store_date_bounds = TRUE

**week_1_start** Date. Sunday of the IBGE first reference week of the month. Only returned if store_date_bounds = TRUE

**week_1_end** Date. Saturday of the IBGE first reference week of the month. Only returned if store_date_bounds = TRUE

**week_2_start** Date. Sunday of the IBGE second reference week of the month. Only returned if store_date_bounds = TRUE

**week_2_end** Date. Saturday of the IBGE second reference week of the month. Only returned if store_date_bounds = TRUE

**week_3_start** Date. Sunday of the IBGE third reference week of the month. Only returned if store_date_bounds = TRUE

**week_3_end** Date. Saturday of the IBGE third reference week of the month. Only returned if store_date_bounds = TRUE

**week_4_start** Date. Sunday of the IBGE fourth reference week of the month. Only returned if store_date_bounds = TRUE

**week_4_end** Date. Saturday of the IBGE fourth reference week of the month. Only returned if store_date_bounds = TRUE

**month_max_upa** Integer. Maximum month position across UPA-V1014 group (for debugging). Only returned if store_date_bounds = TRUE

**month_min_upa** Integer. Minimum month position across UPA-V1014 group (for debugging). Only returned if store_date_bounds = TRUE

**fortnight_max_hh** Integer. Maximum fortnight position within household (for debugging). Only returned if store_date_bounds = TRUE

**fortnight_min_hh** Integer. Minimum fortnight position within household (for debugging). Only returned if store_date_bounds = TRUE

**week_min_hh** Integer. Minimum week position within household (for debugging). Only returned if store_date_bounds = TRUE

**week_max_hh** Integer. Maximum week position within household (for debugging). Only returned if store_date_bounds = TRUE

**ref_month_yyyymm** Integer. Identified reference month in the format YYYYMM, where MM follows the IBGE calendar. 1 <= MM <= 12

**ref_fortnight_yyyyff** Integer. Identified reference fortnight in the format YYYYFF, where FF follows the IBGE calendar. 1 <= FF <= 24

**ref_week_yyyyww** Integer. Identified reference Week in the format YYYYWW, where WW follows the IBGE calendar. 1 <= WW <= 48

**determined_month** Logical. Flags if the month was determined.

**determined_fortnight** Logical. Flags if the fortnight was determined.

**determined_week** Logical. Flags if the week was determined.

## Note

**Nested Identification Hierarchy:**

The algorithm enforces strict nesting by construction:

- Fortnights can ONLY be identified for observations with determined months
- Weeks can ONLY be identified for observations with determined fortnights

This guarantees: `determined_week` => `determined_fortnight` => `determined_month`

**Aggregation Levels:**

The crosswalk aggregates at different levels:

- **Months**: UPA-V1014 level across ALL quarters (PNADC panel design ensures same month position)
- **Fortnights**: Household level within quarter only
- **Weeks**: Household level within quarter only

## See Also

`pnadc_apply_periods` to apply the crosswalk and calibrate weights

## Examples

```
## Not run:
crosswalk <- pnadc_identify_periods(pnadc_stacked)

crosswalk[, .(
  month_rate = mean(determined_month),
  fortnight_rate = mean(determined_fortnight),
  week_rate = mean(determined_week)
)]

crosswalk[determined_fortnight, all(determined_month)]
crosswalk[determined_week, all(determined_fortnight)]

result <- pnadc_apply_periods(pnadc_2023, crosswalk,
                              weight_var = "V1028",
                              anchor = "quarter")

## End(Not run)
```

---

# pnadc_series_starting_points

*Starting Points for SIDRA Series Mensalization*

---

## Description

Pre-computed starting point values (y0) for mensalizing IBGE's rolling quarterly series into exact monthly estimates.

## Usage

```
data(pnadc_series_starting_points)
```

## Format

A data.table with 159 rows and 3 columns:

**series_name** Character. Name of the SIDRA series (53 series)

**mesnotrim** Integer. Month position in quarter (1, 2, or 3)

**y0** Numeric. Starting point value for this series and position

## Details

These starting points were computed from PNADC microdata using the full R package pipeline, ensuring consistency with `compute_starting_points_from_microdata`:

1. Weight calibration via `pnadc_apply_periods`: all months scaled to SIDRA monthly population totals
2. z_ aggregates computed via `compute_z_aggregates` using calibrated `weight_monthly`
3. Starting points computed via `compute_series_starting_points` with CNPJ-aware calibration periods

The calibration period (2013-2019) was chosen because:

- It includes stable pre-pandemic data
- IBGE methodology was consistent during this period
- Sufficient observations for reliable estimates

CNPJ series (empregadorcomcnpj, empregadorsemcnpj, contapropriacomcnpj, contapropriasemcnpj) use calibration period 2016-2019 with cumulative sum starting from October 2015 due to V4019 variable availability.

## Methodology Consistency

The bundled starting points are generated using the same pipeline as `compute_starting_points_from_microdata`, ensuring that users who compute custom starting points will get consistent results.

## When to Use Custom Starting Points

The bundled starting points are suitable for most users. Consider computing custom starting points with `compute_starting_points_from_microdata` if:

- IBGE makes major methodological changes to the PNADC
- You need series not included in the bundled data
- You want to use a different calibration period
- You are working with updated or different microdata

## Source

Computed using `data-raw/regenerate_starting_points_from_microdata.R`

## See Also

`mensalize_sidra_series` which uses this data by default `compute_series_starting_points` for custom calibration

## Examples

```
data(pnadc_series_starting_points)
head(pnadc_series_starting_points)
unique(pnadc_series_starting_points$series_name)
```

---

# validate_pnadc *Validate PNADC Input Data*

---

## Description

Checks that input data has required columns for the specified processing.

## Usage

```
validate_pnadc(data, check_weights = FALSE, stop_on_error = TRUE)
```

## Arguments

| | |
|---|---|
| `data` | A data.frame or data.table with PNADC microdata |
| `check_weights` | Logical. If TRUE, also check for weight-related variables. |
| `stop_on_error` | Logical. If TRUE, stops with an error. If FALSE, returns a validation report list. |

## Details

The function performs the following validations:

- Checks for required columns for reference period identification: `Ano`, `Trimestre`, `UPA`, `V1008`, `V1014`, `V2008`, `V20081`, `V20082`, `V2009`
- Validates year range (2012-2100 for PNADC coverage)
- Validates quarter values (must be 1-4)
- Validates birth day values (must be 1-31 or 99 for unknown)
- Validates birth month values (must be 1-12 or 99 for unknown)
- Warns about unusual ages (outside 0-130 range)
- If `check_weights = TRUE`, also validates weight-related columns: `V1028`, `UF`, `posest`, `posest_sxi`

## Value

If `stop_on_error = TRUE`, returns invisibly if valid or stops with error. If `stop_on_error = FALSE`, returns a list with:

- `valid`: Logical indicating if data passed all validations
- `issues`: Named list of validation issues found (empty if none)
- `n_rows`: Number of rows in input data
- `n_cols`: Number of columns in input data
- `join_keys_available`: Character vector of available join key columns

## See Also

`pnadc_identify_periods` which calls this function internally to validate input data.

## Examples

```
# Minimal valid data (all 9 required columns)
sample_data <- data.frame(
  Ano = 2023L, Trimestre = 1L, UPA = 110000001L,
  V1008 = 1L, V1014 = 1L,
  V2008 = 15L, V20081 = 3L, V20082 = 1990L, V2009 = 33L
)
validate_pnadc(sample_data)

# Data with missing columns returns issues (non-stop mode)
incomplete_data <- data.frame(Ano = 2023L, Trimestre = 1L)
result <- validate_pnadc(incomplete_data, stop_on_error = FALSE)
result$valid    # FALSE
result$issues   # lists missing columns
```

# Index