Package: tidyusmacro (via r-universe)

October 31, 2026

Title Downloading and Cleaning U.S. Macroeconomic Data

Version 0.3.0

Description Utilities to retrieve and tidy U.S. macroeconomic data series from public government data providers. Functions streamline access to series from the Federal Reserve Bank of St. Louis Federal Reserve Economic Data (FRED), the Bureau of Labor Statistics flat files, and the Bureau of Economic Analysis National Income and Product Accounts tables, then return consistent, tidy data frames ready for modeling and graphics. The package includes helpers for date alignment, log-linear projections, and common macro diagnostics, along with convenience plot builders for quick publication-quality charts.

License MIT + file LICENSE

URL https://github.com/mtkonczal/tidyusmacro, https://www.mikekonczal.com/tidyusmacro/

BugReports https://github.com/mtkonczal/tidyusmacro/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

LazyData true

Imports dplyr (>= 1.1.0), ggplot2, httr, magrittr, purrr, readr, rlang, stringi, tidyr

Suggests chromote, curl, knitr, rmarkdown, rvest, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev libssl-dev libx11-dev

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

Date/Publication 2026-09-01 18:16:34 UTC

RemoteUrl https://github.com/mtkonczal/tidyusmacro

RemoteRef HEAD

RemoteSha 16acecb2f411cbc2c375f6d025622f8d793dd5b1

Contents

blsFiles . . . 2
blsSources . . . 3
cesDiffusionIndex . . . 4
dallasTrimPCEcomponents . . . 4
date_breaks_gg . . . 5
date_breaks_n . . . 6
esp_navy . . . 7
esp_pal . . . 8
esp_theme . . . 8
getBLSFiles . . . 10
getCESRevisions . . . 14
getCPIAspects . . . 15
getDallasTrimPCE . . . 18
getFRED . . . 20
getNIPAFiles . . . 21
getPCEInflation . . . 22
getUnrateFRED . . . 24
logLinearProjection . . . 24

Index **26**

`blsFiles` *List every file BLS publishes for a source*

Description

List every file BLS publishes for a source

Usage

```
blsFiles(source, email, data_only = TRUE)
```

Arguments

`source`	A registry name (see `blsSources`) or a bare two-letter BLS prefix (e.g. `"wp"`).
`email`	Contact email, required by BLS in the user agent.
`data_only`	Logical, default `TRUE`: return only `.data.*` files, not lookup tables.

Value

A tibble: `stem` (pass to `getBLSFiles(..., file = )`), `mb`, `modified` (remote mtime), `file` (full BLS name).

Examples

```
## Not run:
  blsFiles("ppi", "your.email@example.com")

## End(Not run)
```

`blsSources` *List available BLS sources*

Description

List available BLS sources

Usage

```
blsSources(tier = NULL, status = "current", pattern = NULL)
```

Arguments

`tier`	Optional integer vector; 1 = release-calendar workhorses, 2 = annual/structural, 3 = specialist, 4 = discontinued.
`status`	`"current"`, `"discontinued"`, or `NULL` for both. Default `"current"`.
`pattern`	Optional regex matched against `name` and `title`.

Value

A tibble: one row per source, with `name`, `prefix`, `file` (default data file), `title`, `frequency`, `tier`, `approx_mb`, `status`, `notes`, `url`.

Examples

```
## Not run:
  blsSources()
  blsSources(tier = 1)
  blsSources(pattern = "price")

## End(Not run)
```

cesDiffusionIndex *cesDiffusionIndex Dataset*

Description

A tibble with 250 rows and 2 columns representing industry codes and corresponding industry titles.

Usage

```
cesDiffusionIndex
```

Format

A tibble with 250 rows and 2 variables:

ces_industry_code A character vector containing the industry codes (e.g., "10-11330000").

ces_industry_title A character vector containing the titles of the industries (e.g., "Logging").

Details

This dataset contains information on different industries, where each row corresponds to an industry defined by its unique code and a descriptive title. It is useful for analyses that require linking industry classifications to descriptive labels.

Source

U.S. Bureau of Labor Statistics (BLS)

Examples

```
# Load the dataset
data(cesDiffusionIndex)
```

dallasTrimPCEcomponents *Dallas Fed Trimmed-Mean PCE component dictionary*

Description

A tibble mapping each of the 177 components used in the Federal Reserve Bank of Dallas's trimmed-mean PCE inflation rate to its corresponding series in BEA NIPA Table 2.4.4U (Fisher price index for personal consumption expenditures by type of product, monthly).

Usage

```
dallasTrimPCEcomponents
```

Format

A tibble with 177 rows and 5 variables:

dallas_idx Integer. Ordinal position in the Dallas Fed tech-notes list.

name Character. Component name as published by the Dallas Fed.

series_code Character. BEA NIPA series code in Table 2.4.4U.

line_no Integer. Line number in BEA Table 2.4.4U.

bea_label Character. Label BEA publishes alongside `series_code`.

Details

Components are organized as durables (1-41), nondurables (42-91), services (92-177), and one NPISH aggregate (178). The 2009 tech notes (Dolmas, "Trimmed Mean PCE Inflation," updated 2022-12-23) list 178 components; BEA combined two of them - Tenant-Occupied Stationary Homes and Tenant Landlord Durables - into a single line of Table 2.4.4U (`IA000629`) when the disaggregated series stopped in December 2001. The mapping reflects that combination, yielding 177 rows; `dallas_idx` runs 1..178 with 94 omitted (93 holds the merged item).

Source

Dolmas, J. (2009, updated 2022-12-23). "PCE Inflation: Technical Note." Federal Reserve Bank of Dallas. BEA NIPA Table 2.4.4U.

References

Atkinson, T., Dolmas, J., & Zarutskie, R. (2026). "Skewness warrants caution as Trimmed Mean PCE inflation eases." Federal Reserve Bank of Dallas, April 16, 2026.

Examples

```
data(dallasTrimPCEcomponents)
head(dallasTrimPCEcomponents)
```

`date_breaks_gg` *Date breaks anchored to last data month (for ggplot)*

Description

Create a breaks function for `scale_x_date()` that always includes the last actual data month and then selects every nth month counting backward.

Usage

```
date_breaks_gg(n = 6, last, decreasing = FALSE)
```

Arguments

n	Integer; keep every n-th month counting backward from last. Default 6.
last	Date; the last (max) date in your data. Required to ensure no break is placed after your actual data.
decreasing	Logical; if TRUE, return breaks in descending order. Default FALSE.

Value

A function usable in scale_x_date(breaks = ...).

See Also

date_breaks_n, which takes the date vector directly and returns a vector of breaks instead of a breaks function.

Examples

```
# Minimal reproducible example (avoid using the name `df`, which masks stats::df)
set.seed(1)
dat <- data.frame(
  date  = seq(as.Date("2023-01-01"), by = "month", length.out = 24),
  value = cumsum(rnorm(24))
)

library(ggplot2)

ggplot(dat, aes(date, value)) +
  geom_line() +
  scale_x_date(
    date_labels = "%b\n%Y",
    breaks = date_breaks_gg(n = 6, last = max(dat$date))
  ) +
  labs(x = NULL, y = NULL)
```

---

date_breaks_n *Create evenly spaced breaks*

Description

Generate a sequence of date breaks for ggplot scales, taking every nth unique date.

Usage

```
date_breaks_n(dates, n = 6, decreasing = TRUE)
```

Arguments

dates	A vector of dates.
n	Integer, keep every n-th date (default = 6).
decreasing	Logical, if TRUE (default) sorts dates in descending order.

Value

A vector of dates suitable for use as ggplot2 axis breaks. With the default `decreasing = TRUE`, the first (most recent) date is always included, so breaks are anchored to the last observation.

See Also

`date_breaks_gg`, which returns a breaks *function* for `scale_x_date()` and clips breaks to the plot limits.

Examples

```
library(ggplot2)
library(dplyr)

df <- tibble(
  date = seq.Date(as.Date("2020-01-01"), as.Date("2025-01-01"), by = "month"),
  value = rnorm(61)
)

ggplot(df, aes(date, value)) +
  geom_line() +
  scale_x_date(breaks = date_breaks_n(df$date, 6))
```

---

esp_navy *ESP Primary Color (Navy)*

Description

The ESP primary color ("Warm Navy", `"#2c3254"`) as a standalone value, convenient for single-series plots: `geom_line(color = esp_navy)`.

Usage

```
esp_navy
```

Format

A named character vector of length 1.

See Also

`esp_pal`, `theme_esp`

Examples

```
esp_navy
```

`esp_pal` *ESP Color Palette*

Description

Named character vector of ESP-branded colors, used by `scale_color_esp` and `scale_fill_esp`.

Usage

```
esp_pal
```

Format

A named character vector of three hex colors:

"Warm Navy" `"#2c3254"`

"Warm Red" `"#ff8361"`

"Soft Green" `"#70ad8f"`

See Also

`esp_navy`, `theme_esp`

Examples

```
esp_pal
esp_pal["Warm Red"]
```

`esp_theme` *ESP Theme and Color Scales*

Description

Custom ggplot2 theme and discrete color/fill scales for Economic Security Project graphics.

Usage

```
theme_esp(base_family = "Public Sans")

scale_color_esp(...)

scale_fill_esp(...)

scale_colour_esp(...)
```

Arguments

base_family	Base font family for the theme. Defaults to "Public Sans"; if the font is not installed, ggplot2 falls back to the default sans font.
...	Passed to the underlying ggplot2 scale functions (scale_color_manual / scale_fill_manual).

Details

theme_esp() builds on theme_minimal with an ESP house style. Note several opinionated defaults: a cream plot background ("#f4f2e4"), no minor gridlines, and **axis titles and the legend removed entirely**. Label lines directly (e.g., with annotate() or geom_text()) or re-enable the legend with theme(legend.position = ...) after theme_esp().

scale_color_esp() and scale_fill_esp() apply the esp_pal palette (three colors, so at most three discrete levels). scale_colour_esp() is an alias.

Value

theme_esp() returns a theme object; the scale functions return ggplot2 scale objects. All are added to a plot with +.

See Also

esp_pal, esp_navy

Examples

```
# Plots are assigned rather than printed: rendering requires the
# "Public Sans" font, which check machines may not have registered.
library(ggplot2)

p1 <- ggplot(economics, aes(date, unemploy / pop)) +
  geom_line(color = esp_navy) +
  labs(
    title = "Unemployment share of population",
    caption = "Source: FRED via ggplot2::economics."
  ) +
  theme_esp()

# Multiple series (palette has 3 colors, so at most 3 levels);
# theme_esp() removes the legend, so label lines directly
dat <- subset(economics_long, variable %in% c("psavert", "uempmed", "unemploy"))
p2 <- ggplot(dat, aes(date, value01, color = variable)) +
  geom_line() +
  scale_color_esp() +
  theme_esp()

# print(p1) or print(p2) to render (needs the font installed)
```

`getBLSFiles` *Download and Process Bureau of Labor Statistics Data*

Description

Downloads and processes data from Bureau of Labor Statistics (BLS) flat files. Supports multiple data sources including CPI, ECI, JOLTS, CPS, CES, and others. The function retrieves the main data file along with associated metadata files, merges them, and returns a tidy tibble ready for analysis.

Usage

```
getBLSFiles(
  data_source,
  email,
  weights = TRUE,
  include_averages = TRUE,
  file = NULL,
  max_mb = 500
)
```

Arguments

`data_source` Character string specifying the BLS data source. Call `blsSources()` for the full, current list with sizes and descriptions. Commonly used values:

- `"cpi"` Consumer Price Index - current data
- `"cpi_w"` CPI, urban wage earners; basis for the Social Security COLA
- `"cpi_chained"` Chained CPI (C-CPI-U); basis for tax bracket indexing
- `"eci"` Employment Cost Index (quarterly)
- `"ecec"` Employer Costs for Employee Compensation (quarterly)
- `"cex"` Consumer Expenditure Survey
- `"jolts"` Job Openings and Labor Turnover Survey
- `"cps"` Current Population Survey
- `"ces"` Current Employment Statistics - all series
- `"ces_allemp"` Current Employment Statistics - all employees, seasonally adjusted
- `"ces_total"` Current Employment Statistics - total nonfarm employment
- `"averageprice"` Average price data - current
- `"food"` Average price data - food items
- `"ppi"` Producer Price Index, commodity
- `"ppi_industry"` Producer Price Index, industry and product
- `"import_export"` Import and export price indexes
- `"productivity"` Major sector productivity and unit labor costs
- `"laus"` Local Area Unemployment Statistics (was `"su"`)

"sae" State and Area Employment, Hours, and Earnings (was "se")

"se" and "su" still work but are deprecated: "su" is a real BLS prefix (chained CPI), which the old alias was squatting on, so it could not be reused once chained CPI was added. They warn and redirect to "sae"/"laus".

email — Character string with your email address. Required by BLS for identifying API users. Set as the HTTP User-Agent header.

weights — Logical; for data_source = "cpi" only. When TRUE (the default), also downloads cu.aspect and attaches monthly relative importance plus BLS's own published contributions. Ignored for every other data source. Set to FALSE to skip the extra ~31 MB download.

include_averages — Logical, default TRUE. BLS period codes include not just monthly/quarterly observations but computed averages: M13 (annual average), S01/S02 (half-year average), S03 (annual average), and Q05 (annual average). These are flagged via the is_average column rather than dropped, because for some sources (CEX publishes only A01) they are the only rows that exist. Set FALSE to drop them, e.g. before a group_by(date) across many series where an average row would otherwise land on the same December date as that year's real December observation.

file — Character, optional. Picks a non-default data file within the survey, e.g. file = "data.21.Aggregates" for PPI's FD-ID aggregates. Call blsFiles(data_source, email) to see what exists. Required for the discontinued (tier 4) surveys, which have no default file.

max_mb — Numeric, default 500. Refuse to download a data file larger than this without an explicit override (e.g. osh_characteristics is 2.9 GB). Set Inf to disable.

Details

The function constructs URLs to BLS flat files at https://download.bls.gov/pub/time.series/, downloads the series metadata and auxiliary lookup tables, then downloads and merges the main data file.

Value

A tibble containing the merged data with columns for:

series_id — Unique identifier for each data series

date — Observation date. See the "Period parsing" section below.

freq — One of "monthly", "quarterly", "semiannual", or "annual", from the BLS period code.

is_average — TRUE for rows BLS computed as an average or annual aggregate (period codes M13, S01, S02, S03, Q05) rather than observed in that period.

value — Numeric data value

... — Additional metadata columns vary by data source (e.g., item codes, industry codes, area codes)

For CPI with weights = TRUE, four further columns:

weight	Relative importance, in percent of all items, on the base month for the 1-month change ending in this observation month. This is the weight for a 1-month contribution; do not lag it. See the dating note below.
weight_12mo	Relative importance on the base month for the 12-month change ending in this observation month; the weight for a 12-month contribution
effect_1m	BLS's own published effect on the 1-month all items change, in percentage points. Seasonally adjusted rows only.
effect_12m	BLS's own published effect on the 12-month all items change, in percentage points. Not seasonally adjusted rows only.

Period parsing

Prior to this version, date was computed as substr(period, 2, 3) for every source except ECI. That is correct for monthly (M01-M12) and, via a special case, for ECI's quarterly (Q01-Q04) codes, but every source can also carry BLS-computed average rows on other period codes, and the old parser mis-stamped them: S01/S02 (half-year averages) became January/February, S03 became March, and M13 (annual average) became a silent NA. Verified on cu.data.1.AllItems (2026-08-19): 26\ every period code (see is_average above) and gives every row a correct date.

The month is the *end* of the period, matching the pre-existing ECI convention (Q01 is March). Annual and half-year rows land in the last month they cover: M13/S03/Q05/A01 in December, S01 in June.

The *day* separates observed values from computed averages. An observed value is dated the **first** of its month, matching every other date this package returns (getFRED, getNIPAFiles). A computed average is dated the **last day** of its terminal month: 2024-12-31 for M13, 2024-06-30 for S01.

That rule exists because the month alone cannot separate them. There is no month an annual average can occupy that some observed month does not already own, so under a uniform first-of-month rule the 2024 annual average and the real December 2024 observation are the same Date, and a group_by(date) across many series double-counts silently. With the day rule that is impossible rather than merely documented.

Averages still share a date with each *other*: M13, S02 and S03 all land on December 31. They are all averages, so is_average or freq separates them and no observed value is ever contaminated. The full key is (series_id, date, freq), not date alone.

CPI relative importance

Relative importance comes from cu.aspect (aspect type "I"), which BLS restamps with every CPI release. Three things about the join are worth knowing:

- It is keyed on area_code + item_code + date, not on series_id. BLS publishes relative importance only on the not seasonally adjusted series (CUUR...), but the weight describes the item, not the adjustment, so joining on series_id would return all NA for seasonally adjusted work. Codes rather than names, because BLS renames items and the codes are stable.
- Coverage is U.S. city average (area_code == "0000") from March 2012 forward. Outside that window weight is NA rather than back-filled: an imputed weight that looks like a real one is worse than a missing value. See the BLS relative importance archive for a pre-2012 backfill.

- A row of `cu.aspect` stamped month *t* carries the relative importance BLS labels month *t-1*. This is the one thing about the file that reliably produces off-by-one errors, so it is worth stating twice: the weight you want for the change *ending* in month *t* is the row dated *t*, not a lag of it. Verified against the June 2026 release, where the "Relative importance May 2026" column of Tables 6 and 7 matches the 2026-06-01 rows for all 307 items exactly and the 2026-05-01 rows for only 43. The same shift is why BLS's published "Relative importance, December YYYY" table is the **January YYYY+1** row.
- Accordingly `weight` is the row dated *t* and needs no lag, and `weight_12mo` is the row dated *t-11* – eleven months back, because the RI labeled *t-12* lives in the *t-11* row.
- Both weight columns are joined on a month index, never by row position. BLS omits rows entirely for intermittently priced items rather than writing NA, so a positional lag borrows the wrong month's weight without warning.

Contributions

`effect_1m` and `effect_12m` are BLS's own decomposition, and they equal the "effect on All Items" columns of news release Tables 6 and 7 exactly. Use them for anything BLS publishes.

`weight` and `weight_12mo` are for aggregations BLS does not publish. For a 12-month contribution, `weight_12mo * (value / lag12(value) - 1)` on the NSA series is a good approximation. For a 1-month contribution on the *seasonally adjusted* series, note that relative importance is defined on the NSA index and has to be rescaled by the item's seasonal factor relative to all items before it will reproduce BLS's number; see `getCPIAspects` for the exact formula.

In both cases lag by calendar month, not row position.

See Also

`getCPIAspects` for the other CPI aspect types: BLS's own contribution decomposition, median standard errors, and seasonal factors.

Examples

```
## Not run:
  # Download CPI data with monthly relative importance attached
  cpi_data <- getBLSFiles("cpi", "your.email@example.com")

  # Skip the weights download
  cpi_fast <- getBLSFiles("cpi", "your.email@example.com", weights = FALSE)

  # Download JOLTS data
  jolts_data <- getBLSFiles("jolts", "your.email@example.com")

## End(Not run)
```

`getCESRevisions` *Download CES Monthly Jobs Revisions*

Description

Scrapes the Bureau of Labor Statistics table of monthly Current Employment Statistics (CES) revisions from https://www.bls.gov/web/empsit/cesnaicsrev.htm and returns a tidy tibble of first, second, and third prints of the over-the-month change in total nonfarm employment (1979-present), seasonally adjusted and not seasonally adjusted, along with the revision deltas between prints.

Usage

```
getCESRevisions(
  source_url = "https://www.bls.gov/web/empsit/cesnaicsrev.htm",
  timeout = 60
)
```

Arguments

`source_url`	Character string with the BLS revisions page URL. Defaults to the CES NAICS revisions page; override only if BLS moves the page.
`timeout`	Numeric. Maximum seconds to wait for the page to load in headless Chrome before failing. Default 60.

Details

The BLS revisions page sits behind Akamai bot detection that rejects requests from R's HTTP stack (`httr`/`curl`) based on their TLS fingerprint, regardless of the User-Agent header. This function therefore fetches the page through a headless Chrome browser via the **chromote** package, which requires a local installation of Google Chrome (or another Chromium-based browser). Both **chromote** and **rvest** must be installed to use this function.

Each calendar year on the page is published as its own table with a three-row header (adjustment status, measure, print). Tables are parsed with `rvest::html_table()`, headers are flattened, and target columns are located by pattern so that reordering or added columns on the BLS page do not silently misalign values. If the same year-month appears in more than one table, the most complete row is kept.

All values are over-the-month changes in thousands of jobs. Recent months will have `NA` second or third prints until those estimates are published; revision columns are `NA` until both prints exist.

Value

A tibble with one row per month and columns:

`date`	Observation month (first of month).
`year`	Calendar year (integer).

month	Three-letter month abbreviation.
month_num	Month number 1-12 (integer).
sa_1st, sa_2nd, sa_3rd	Seasonally adjusted first, second, and third prints of the over-the-month employment change (thousands).
sa_rev_2nd_minus_1st, sa_rev_3rd_minus_2nd, sa_rev_3rd_minus_1st	Seasonally adjusted revisions between prints (thousands).
nsa_1st, nsa_2nd, nsa_3rd	Not seasonally adjusted prints (thousands).
nsa_rev_2nd_minus_1st, nsa_rev_3rd_minus_2nd, nsa_rev_3rd_minus_1st	Not seasonally adjusted revisions (thousands).
source_url	URL the data was scraped from.
scraped_at	UTC timestamp of the download (vintage).

Examples

```
## Not run:
  revisions <- getCESRevisions()

  # Average absolute seasonally adjusted revision, first to third print:
  mean(abs(revisions$sa_rev_3rd_minus_1st), na.rm = TRUE)

## End(Not run)
```

---

getCPIAspects *Download CPI Metadata (Aspect) Files from BLS*

Description

Downloads the BLS CPI "aspect" flat file, which carries the monthly metadata that accompanies each published CPI series: relative importance (weights), BLS's own contribution-to-the-all-items-change decomposition, median standard errors, seasonal factors, and published percent changes.

Usage

```
getCPIAspects(email, survey = c("cu", "cw"), aspect_type = NULL)
```

Arguments

email	Character string with your email address. Required by BLS for identifying users; set as the HTTP User-Agent header.
survey	Character string, either "cu" (CPI-U, the default) or "cw" (CPI-W).
aspect_type	Optional character vector of aspect codes to keep (e.g. "I" for relative importance). NULL (the default) returns every aspect type.

Details

The aspect file lives at `https://download.bls.gov/pub/time.series/cu/cu.aspect` and is restamped with every CPI release. It is not described in `cu.txt` (that file was last revised in February 2018; the aspect files were added in November 2024); the documentation is on a separate BLS fact sheet.

Value

A tibble with one row per series/month/aspect_type and columns:

`series_id` Full BLS series identifier

`area_code`, `item_code`, `seasonal`, `periodicity_code`
Components parsed out of `series_id`

`year`, `period`, `date`
Observation month. Dates come from the shared parser in `bls_parse_period()`, the same one `getBLSFiles()` uses.

`freq`, `is_average`
Frequency implied by the BLS period code, and whether the row is a BLS-computed average. Every aspect type BLS currently publishes is monthly (verified live 2026-08-31), so `is_average` is presently `FALSE` throughout.

`aspect_type` Aspect code (see Details)

`value` Published value as a character string, exactly as distributed

`value_num` Numeric version of `value`; `NA` for the text aspect types `H1` and `HC`

`footnote_codes` BLS footnote codes, if any

Dating convention

A row stamped month *t* carries the relative importance BLS labels month *t-1*. That is the weight base for the *t-1* to *t* change, so the row you want for a change ending in month *t* is the row dated *t* – not a lag of it.

Verified against the June 2026 news release: the "Relative importance May 2026" column in Tables 6 and 7 matches the rows dated 2026-06-01 for all 307 items exactly, and matches the rows dated 2026-05-01 for only 43 of them. The same shift explains why BLS's published "Relative importance, December YYYY" table is the **January YYYY+1** row of this file.

Aspect types, and the seasonal-adjustment domain each one is published on:

`I` Relative importance, monthly. NSA series only (`CUUR`/`CWUR`), March 2012 forward.

`I1` End-of-year relative importance. NSA only, Dec 2020 forward.

`F` Seasonal factor. SA series (`CUSR`).

`W1` Effect on the 1-month all items change, in percentage points. Published on the *seasonally adjusted* series.

`WC` Effect on the 12-month all items change, in percentage points. Published on the *not seasonally adjusted* series.

`V1`, `VC` The percent change *at the reference month named by* `H1`/`HC`, not the current month's change. Read them together with `H1`/`HC`: they are the two right-hand columns of news release Tables 6 and 7 ("Largest (L) or Smallest (S) change since: Date / Percent change"). The current month's percent change is not in this file; compute it from the index.

`M1`, `MC` Median standard error of the 1-month (SA) and 12-month (NSA) percent change.

`H1`, `HC` Text notes flagging largest/smallest change since a reference date. Character, not numeric.

Note that `W1`/`V1`/`M1` attach to the SA series and `WC`/`VC`/`MC` to the NSA series, matching BLS's convention of reporting 1-month changes seasonally adjusted and 12-month changes not seasonally adjusted. Join those on the full `series_id`. Relative importance (`I`) is the exception: it is defined only on the NSA series but describes the item, so it applies to the SA series too and should be joined on `area_code` + `item_code` + `date`. This is what `getBLSFiles("cpi", ...)` does.

Reproducing the published effect columns

`W1` and `WC` are BLS's own contribution decomposition, and they equal the "effect on All Items" columns of Tables 6 and 7 exactly (verified for June 2026: 269 of 269 and 306 of 306 items, zero deviation). Prefer them to rolling your own.

If you do need to roll your own – for a custom aggregation BLS does not publish – the 1-month effect is *not* relative importance times the seasonally adjusted percent change. Relative importance is defined on the NSA index, so it has to be put on an SA footing first:

$$W1_{i,t} = I_{i,t} \times \frac{SA_{i,t-1}/NSA_{i,t-1}}{SA_{all,t-1}/NSA_{all,t-1}} \times \frac{SA_{i,t}/SA_{i,t-1} - 1}{1} \times 100$$

That reproduces `W1` exactly (270 of 270 items in June 2026). Dropping the seasonal-factor ratio costs 0.018 percentage points on gasoline and 0.008 on energy – small, but large enough to change a rounded headline contribution. There is no equally clean reconstruction of `WC`: chaining twelve monthly NSA effects lands within 0.027 percentage points, which is why the recommendation is to use BLS's value.

See Also

`getBLSFiles`, which attaches relative importance to CPI index values directly.

Examples

```
## Not run:
  # Everything
  aspects <- getCPIAspects("your.email@example.com")

  # BLS's own contribution decomposition, to check your own against
  effects <- getCPIAspects("your.email@example.com", aspect_type = c("W1", "WC"))

## End(Not run)
```

`getDallasTrimPCE` *Build the Dallas Fed trimmed-mean PCE component panel*

Description

Returns a long tibble with the raw inputs to the Federal Reserve Bank of Dallas's trimmed-mean PCE inflation rate: monthly Fisher price index, nominal expenditure, real quantity, monthly price change, the Fisher (t-1, t) expenditure-share weight, and a flag indicating whether the component was trimmed in that month and on which tail. Users can replicate the trimmed-mean rate by collapsing this tibble to kept (non-trimmed) components each month and taking the weight-renormalized weighted mean of price changes.

Usage

```
getDallasTrimPCE(
  frequency = "M",
  NIPA_data = NULL,
  alpha = 0.24,
  beta = 0.31,
  components = NULL
)
```

Arguments

`frequency`	Character. Frequency code passed to `getNIPAFiles`. Defaults to `"M"` (monthly). Currently the trimmed-mean panel is only meaningful at monthly frequency.
`NIPA_data`	Optional pre-loaded NIPA tibble from `getNIPAFiles()`. If `NULL`, the function downloads it.
`alpha`	Numeric in [0, 1]. Lower-tail trim share. Default `0.24`, the Dallas Fed published value.
`beta`	Numeric in [0, 1]. Upper-tail trim share. Default `0.31`.
`components`	Optional override for the component dictionary. Must be a tibble with columns `dallas_idx`, `name`, `series_code`, `line_no`. Defaults to the packaged `dallasTrimPCEcomponents` (177 components).

Details

Weights are Fisher-style: an unweighted average of the expenditure share evaluated at base prices `P[t-1]` with quantities `Q[t-1]` and `Q[t]`, renormalized to sum to 1 within each month. Real quantity is computed as `nominal / price` from BEA NIPA Tables 2.4.5U and 2.4.4U respectively (equivalent to Table 2.4.6U per the Dallas Fed's MATLAB note, but available across the full sample without chained-dollar gaps).

Trim assignment is the simple rank-based version: components are sorted within each month by `price_change`, cumulative weight is accumulated, and components whose running cumulative weight is below `alpha` are flagged `"lower"`, while components whose cumulative weight before

adding their own contribution is at or above `1 - beta` are flagged `"upper"`. Boundary components that straddle either threshold are kept (treated as interior). The Dallas Fed's exact fractional-boundary handling enters the rate calculation itself, not this panel-builder; the resulting headline rate matches the Dallas Fed series to within a few basis points.

Months without full cross-sectional coverage (i.e., any component missing this month or last) have `weight`, `is_trimmed`, and `trim_side` set to `NA`.

Value

A `tbl_df` with one row per (date, component) and columns:

date Month observation date.

dallas_idx Component ordinal in the Dallas tech notes (1..178, 94 omitted).

name Dallas Fed component name.

series_code BEA NIPA series code (Table 2.4.4U).

line_no BEA NIPA Table 2.4.4U line number.

price Fisher price index (Table 2.4.4U).

nominal Current-dollar outlay (Table 2.4.5U).

quantity Real quantity (`nominal / price`).

price_change Period-over-period fractional change in `price`. `NA` for the first observation per component.

weight Fisher (t-1, t) expenditure-share weight, renormalized to sum to 1 within each full-coverage month. `NA` otherwise.

is_trimmed Logical. `TRUE` if the component is in either tail this month and so dropped from the trimmed mean. `NA` when the month lacks full coverage.

trim_side Character. `"lower"` or `"upper"` when trimmed; `NA` when kept (interior) or coverage incomplete.

References

Dolmas, J. (2005). "Trimmed Mean PCE Inflation." Federal Reserve Bank of Dallas Working Paper 0506.

Dolmas, J. (2009, updated 2022-12-23). "PCE Inflation: Technical Note." Federal Reserve Bank of Dallas.

Atkinson, T., Dolmas, J., & Zarutskie, R. (2026). "Skewness warrants caution as Trimmed Mean PCE inflation eases." Federal Reserve Bank of Dallas, April 16, 2026.

See Also

`dallasTrimPCEcomponents`, `getNIPAFiles`

Examples

```
## Not run:
  # Default 24/31 Dallas Fed trim
  panel <- getDallasTrimPCE()

  # Replicate the monthly trimmed-mean rate (kept components, renormalized
  # to sum to 1):
  library(dplyr)
  tm_rate <- panel |>
    dplyr::filter(!is_trimmed) |>
    dplyr::group_by(date) |>
    dplyr::summarize(
      rate = sum(price_change * weight) / sum(weight),
      .groups = "drop"
    )

  # What got trimmed in the latest month, by tail and weight:
  panel |>
    dplyr::filter(date == max(date), is_trimmed) |>
    dplyr::arrange(trim_side, dplyr::desc(weight)) |>
    dplyr::select(name, trim_side, weight, price_change)

## End(Not run)
```

`getFRED` *Download and Merge FRED Series*

Description

A flexible wrapper that downloads one or more data series from the St. Louis Fed (FRED) API, optionally computes one-period percentage changes, and merges them into a tidy tibble keyed by `date`.

Usage

```
getFRED(..., keep_all = TRUE, rename_variables = NULL, lagged = NULL)
```

Arguments

`...` One or more FRED series IDs. Each element may be either

Unnamed character string The raw FRED ticker; column keeps the lowercase ticker name, e.g.\ `"UNRATE"`.

Named character string The value is the FRED ticker and the name becomes the column label, e.g.\ `payroll = "PAYEMS"`.

You may also pass a single character vector (named or unnamed) for compatibility with older code.

keep_all — Logical. TRUE (default) performs a full join that keeps all dates across series; FALSE performs an inner join.

rename_variables — Optional character vector of new column names (one per series), retained for backward compatibility. Supply *either* this argument *or* names in ..., not both.

lagged — Logical scalar or logical vector. If TRUE (or the corresponding element is TRUE), the series is replaced by its one-period percentage change $(x_t/x_{t-1}) - 1$. Recycled to match the number of series if length 1.

Details

You may supply the series in two ways:

- **Natural "..." style**: `getFRED(unrate = "UNRATE", payroll = "PAYEMS")`. Named arguments give friendly column names; unnamed arguments keep the (lower-case) ticker as the column name.
- **Legacy style**: pass a single (optionally named) character vector—e.g.\`c(unrate = "UNRATE", payroll = "PAYEMS")`—and/or use the `rename_variables=` argument. This remains supported for backward compatibility.

If you provide names in ... *and* a non-NULL `rename_variables` vector, the function stops and prompts you to choose a single naming method.

Value

A tibble with a `date` column and one column per requested series.

Examples

```
## Not run:
# New interface
getFRED(unrate = "UNRATE", payroll = "PAYEMS")

# Multiple unnamed series (columns become 'unrate' and 'payems')
getFRED("UNRATE", "PAYEMS")


## End(Not run)
```

---

getNIPAFiles *Download and Process BEA NIPA Files with Fast Row Expansion*

Description

This function downloads and processes National Income and Product Accounts (NIPA) data files from the BEA website. It reads the necessary register files, formats the date column, and then uses the fast stringi functions together with tidyr's unnest() to split the combined `TableId:LineNo` field into separate rows and columns. Finally, it merges the datasets.

Usage

```
getNIPAFiles(
  location = "https://apps.bea.gov/national/Release/TXT/",
  type = "Q"
)
```

Arguments

location	The URL or path where the BEA files are located. Default: "https://apps.bea.gov/national/Release/TXT/".
type	A character string indicating the type of data to load. For example, "Q" for quarterly or "M" for monthly data. Default is "Q".

Value

A tibble with one row per (series, period, table line). Because a NIPA series can appear on multiple table lines, series-periods are duplicated across rows; filter on `TableId` (and `LineNo`) before analysis. Key columns:

SeriesCode	BEA series mnemonic (from `SeriesRegister.txt`).
Period	Raw BEA period string, e.g. `"2024Q1"` or `"2024M01"`.
date	Period as a `Date` (quarters mapped to their final month).
Value	Numeric data value.
TableId	NIPA table identifier, e.g. `"U20404"`.
LineNo	Numeric line number within the table.
...	Additional series metadata from `SeriesRegister.txt` (including `SeriesLabel`) and table metadata from `TablesRegister.txt`.

Examples

```
## Not run:
  nipadata <- getNIPAFiles(type = "Q")

## End(Not run)
```

`getPCEInflation` *Load and Process Personal Consumption Expenditures (PCE) Inflation Data*

Description

Downloads and processes BEA NIPA data to compute Personal Consumption Expenditures (PCE) price indices with weights and growth measures. This is the Federal Reserve's preferred inflation measure.

Usage

```
getPCEInflation(frequency = "M", NIPA_data = NULL)
```

Arguments

frequency	Character string indicating the frequency of the data: "M" (monthly, the default) or "Q" (quarterly). Also sets the compounding used to annualize WDataValue_P1a (12 for monthly, 4 for quarterly).
NIPA_data	Optional data frame. If provided, it will be used as the raw NIPA dataset instead of loading fresh data with getNIPAFiles(). Make sure frequency matches the frequency of the supplied data, since it determines the annualization exponent.

Details

The function performs the following steps:

1. Loads NIPA data using getNIPAFiles (or uses pre-loaded data).
2. Extracts total PCE from table "U20405" (series code "DPCERC").
3. Computes PCE component weights as the nominal consumption share (component value divided by total PCE).
4. Extracts quantity indices from table "U20403".
5. Loads price indices from table "U20404", joins weights and quantities, and calculates period-over-period growth measures.

Value

A tibble with one row per (date, PCE component), containing the columns from getNIPAFiles for price-index table "U20404" (including date, SeriesLabel, and the price index in Value), plus:

PCEweight	Nominal consumption share: component spending divided by total PCE (both from table "U20405").
quantity	Real quantity index from table "U20403".
DataValue_P1	1-period percent change in the price index (decimal).
DataValue_P3	3-period percent change (decimal).
DataValue_P6	6-period percent change (decimal).
WDataValue_P1	Contribution to 1-period PCE inflation: DataValue_P1 times the lagged PCEweight.
WDataValue_P1a	WDataValue_P1 annualized by compounding over the periods per year implied by frequency: (1 + x)^12 - 1 for monthly, (1 + x)^4 - 1 for quarterly.

Examples

```
## Not run:
  # Load monthly PCE data
  pce_data <- getPCEInflation("M")

## End(Not run)
```

getUnrateFRED *Get Full Unemployment Rate from FRED*

Description

Downloads the civilian unemployment level and labor force level from FRED, and calculates the unemployment rate as unemploy_level/lf_level.

Usage

```
getUnrateFRED()
```

Value

A tibble with columns:

`date`	Observation date
`unemploy_level`	Civilian unemployment level (in thousands)
`lf_level`	Civilian labor force level (in thousands)
`full_unrate`	Unemployment rate (decimal)

Examples

```
## Not run:
  getUnrateFRED()

## End(Not run)
```

logLinearProjection *Log-Linear Projection (data-masked, dplyr-native)*

Description

Fits a log-linear trend `log(value) ~ t` on a calibration window and projects it for rows on/after `start_date`. Designed for use inside dplyr verbs (no need to pass `.`).

Usage

```
logLinearProjection(
  date,
  value,
  start_date,
  end_date,
  group = NULL,
  data = NULL
)
```

Arguments

date	Bare column name for the date variable (coercible to Date).
value	Bare column name for the positive numeric series to project.
start_date	Date or string coercible to Date; start of calibration.
end_date	Date or string coercible to Date; end of calibration.
group	Optional bare column name to group by before projecting.
data	Optional data frame. If omitted, uses the columns visible in the current data mask (e.g., inside mutate()) via dplyr::pick().

Value

A numeric vector projection aligned to the input rows; NA before start_date. Respects grouping if group is supplied.

Index

∗ **datasets**
cesDiffusionIndex, 4
dallasTrimPCEcomponents, 4

blsFiles, 2, *11*
blsSources, *2*, 3, *10*

cesDiffusionIndex, 4

dallasTrimPCEcomponents, 4, *18, 19*
date_breaks_gg, 5, *7*
date_breaks_n, 6, *6*

esp_navy, 7, *8, 9*
esp_pal, *7*, 8, *9*
esp_theme, 8

getBLSFiles, 10, *17*
getCESRevisions, 14
getCPIAspects, *13*, 15
getDallasTrimPCE, 18
getFRED, *12*, 20
getNIPAFiles, *12*, *18, 19*, 21, *23*
getPCEInflation, 22
getUnrateFRED, 24

logLinearProjection, 24

scale_color_esp, *8*
scale_color_esp *(*esp_theme*)*, 8
scale_color_manual, *9*
scale_colour_esp *(*esp_theme*)*, 8
scale_fill_esp, *8*
scale_fill_esp *(*esp_theme*)*, 8
scale_fill_manual, *9*

theme, *9*
theme_esp, *7, 8*
theme_esp *(*esp_theme*)*, 8
theme_minimal, *9*