Package 'indexDesignWindows'

December 13, 2025

Type Package

Title Replication Package: Redesigning Historical Windows in Index-Based Insurance

Version 0.0.0.9000

Author Francis Tsiboe [aut, cre] (<https://orcid.org/0000-0001-5984-1072>)

Maintainer Francis Tsiboe <ftsiboe@hotmail.com>

Creator Francis Tsiboe

Description Replication package for a study evaluating alternative historical window designs used to construct the Pasture, Rangeland, and Forage (PRF) rainfall index under the U.S. Federal Crop Insurance Program (FCIP). The package systematically compares index designs based on varying lengths of historical climate data to assess implications for index stability, spatial equity, indemnity accuracy, and policy performance. It provides reproducible workflows, pre-processed outputs, and visualization tools to support robustness analysis of index-based insurance products in both U.S. and international contexts.

License GPL-3 + file LICENSE

URL https://github.com/ftsiboe/indexDesignWindows

BugReports https://github.com/ftsiboe/indexDesignWindows/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports data.table, ggplot2

Remotes github::ftsiboe/rfcipDemand

Suggests dplyr, tidyr, knitr, rmarkdown, withr, stats, rfcipDemand, piggyback, urbnmapr, sf, classInt, testthat (>= 3.0.0)

LazyData true

Cite-us If you find it useful, please consider staring the repository and citing the following studies
- Tsiboe, F. and Turner, D. (2025). ``Incorporating buy-up price loss coverage into the United States farm safety net." Applied Economic Perspectives and Policy.
- Tsiboe, F., et al. (2025). ``Risk reduction impacts of crop insurance in the United States." Applied Economic Perspectives and Policy.
- Gaku, S. and Tsiboe, F. (2024). Evaluation of alternative farm safety net program combination strategies. Agricultural Finance Review.

Contents

`add_break_categories` *Add Categorical Breaks to a Numeric Variable*

Description

Classifies a numeric variable into categorical intervals using either Fisher-Jenks natural breaks (via **classInt**) or user-specified break points, and appends the resulting factor variable to the input data.

Usage

```
add_break_categories(
  data,
  variable,
  break_n = NULL,
  break_levels = NULL,
  break_labels = NULL
)
```

Arguments

`data`	A data frame containing the variable to be classified.
`variable`	Character scalar giving the name of the numeric column in `data` to classify.
`break_n`	Integer number of classes to compute when using Fisher-Jenks breaks. Required when `break_levels` is `NULL`.
`break_levels`	Optional numeric vector of break points. If supplied, these are used directly and `break_n` is ignored.
`break_labels`	Optional character vector of labels to use for the resulting factor. If `NULL`, labels are generated from `break_levels`.

Details

The function supports two main workflows:

- **Automatic breaks**: If `break_levels` is `NULL`, Fisher-Jenks natural breaks are computed using `classInt::classIntervals()` with `style = "fisher"`. In this case, `break_n` must be supplied.
- **Manual breaks**: If `break_levels` is provided, these are used directly and `break_n` is ignored. If `break_labels` is `NULL`, labels are generated from `break_levels` in the form `"a - b"` (two decimal places).

Intervals are constructed with `include.lowest = TRUE` and `right = FALSE`, so they are left-closed and right-open.

Value

The original `data` with an additional factor column named `<variable>_cat` containing the categorical breaks.

`compute_balance` *Distributional balance: one basis vs multiple comparison variables (long format)*

Description

For a given **basis** numeric column (`col_x`) and one or more **comparison** numeric columns (`col_y`), this function computes a rich set of diagnostics describing bias, scale, agreement, and distributional differences between the basis and each comparison variable.

Results are returned in **long format** with one row per group (defined by `by`) and per comparison variable (`y_level`).

Metrics include:

- **Sample size & moments**
  - `n`: number of non-missing pairs
  - `mean_x`, `mean_y`
  - `var_x`, `var_y`
  - `sd_x`, `sd_y`
  - `cv_x`, `cv_y` (coefficients of variation)
- **Bias & scale**
  - `mean_bias` = mean_x - mean_y
  - `mae` = mean(|x - y|)
  - `rmse` = sqrt(mean((x - y)^2))
  - `smape` = 2 * mean(|x - y| / (|x| + |y|)), with safe handling when denominator is zero
  - `ratio_means` = mean_x / mean_y
  - `cv_ratio` = cv_x / cv_y
  - `mpd` = mean percentage difference:

$$(((\bar{x} - \bar{y})/\bar{y}) - 1) * 100$$

- **Balance / distribution comparison**
  - `msd` = mean standardized difference: (mean_x - mean_y) / sqrt((sd_x^2 + sd_y^2) / 2)
  - `var_ratio` = var_x / var_y
  - `ks_stat` = Kolmogorov-Smirnov D-statistic (two-sample)
  - `emd` = 1D Wasserstein / Earth Mover's Distance, approximated as mean absolute difference between quantiles of x and y
  - `hellinger` = Hellinger distance computed from common histograms
  - `pvalue_mean` = p-value for equality of means (paired t-test)
  - `pvalue_ks` = p-value from Kolmogorov-Smirnov test
  - `pvalue_var` = p-value from F-test of equal variances
  - `pvalue_levene` = p-value from Levene test (mean-based)
  - `pvalue_brown_forsythe` = p-value from Brown-Forsythe test
  - `pvalue_kruskal_wallis` = p-value from Kruskal-Wallis test
- **Zero-mass / structural zeros**
  - `zero_share_x` = proportion of observations with x == 0
  - `zero_share_y` = proportion of observations with y == 0
  - `zero_share_diff` = zero_share_x - zero_share_y
  - `pvalue_zero_mcnemar` = p-value from McNemar test on the paired zero/non-zero indicators
- **Bounded-scale / logit-space (for (0, 1) variables)**
  - `mean_logit_x`, `mean_logit_y` = means of logit-transformed x and y
  - `mean_logit_diff` = mean(logit_x - logit_y)
  - `rmse_logit` = sqrt(mean((logit_x - logit_y)^2))
  - `pvalue_mean_logit` = p-value from paired t-test on logit_x vs logit_y

  These logit-based metrics are computed only if all non-missing x and y are in the interval (0, 1). Otherwise, they are returned as NA.
- **Agreement / association**
  - `pearson_cor` = Pearson correlation
  - `spearman_cor` = Spearman rank correlation
  - `kendall_cor` = Kendall's tau
  - `ccc` = Lin's Concordance Correlation Coefficient
  - `pvalue_pearson_cor`
  - `pvalue_spearman_cor`
  - `pvalue_kendall_cor`
- **Quantile / tail behavior**
  - Quantiles at p = 0.10, 0.25, 0.50, 0.75, 0.90 for x and y: `q10_x`, `q25_x`, `q50_x`, `q75_x`, `q90_x` `q10_y`, `q25_y`, `q50_y`, `q75_y`, `q90_y`
  - Differences: `q10_diff`, `q25_diff`, `q50_diff`, `q75_diff`, `q90_diff` computed as (q_x - q_y)

Usage

```
compute_balance(dt, col_x, col_y, by = NULL)
```

Arguments

dt	A data.frame or data.table.
col_x	Character scalar. Name of the **basis** numeric column.
col_y	Character vector. Names of one or more **comparison** numeric columns. The basis variable col_x is always compared against each variable in col_y.
by	Optional character vector of grouping variables. If NULL (default), all rows are treated as one group.

Value

A data.table in **long format** with columns: c(by, "y_level", "n", "mean_x", "mean_y", "var_x", "var_y", "sd_x", "sd_y", "cv_x", "cv_y", "cv_ratio", "mean_bias", "mae", "rmse", "smape", "ratio_means", "mpd", "msd", "var_ratio", "ks_stat", "emd", "hellinger", "pvalue_mean", "pvalue_ks", "pvalue_var", "pvalue_levene", "pvalue_brown_forsythe", "pvalue_kruskal_wallis", "zero_share_x", "zero_share_y", "zero_share_diff", "pvalue_zero_mcnemar", "mean_logit_x", "mean_logit_y", "mean_logit_diff", "rmse_logit", "pvalue_mean_logit", "pearson_cor", "spearman_cor", "kendall_cor", "ccc", "pvalue_pearson_cor", "pvalue_spearman_cor", "pvalue_kendall_cor", "q10_x", "q25_x", "q50_x", "q75_x", "q90_x", "q10_y", "q25_y", "q50_y", "q75_y", "q90_y", "q10_diff", "q25_diff", "q50_diff", "q75_diff", "q90_diff").

---

ers_theme *ERS Theme*

Description

ERS Theme

Usage

```
ers_theme()
```

Source

coppied from https://github.com/USDA-REE-ERS/MTED-Theme on 08/01/2025

Examples

```
ggplot2::ggplot() + ers_theme()
```

get_price_indices *Build a price-received deflator (PPIPR) series relative to* current_year

Description

Constructs a table used to deflate nominal FCIP monetary amounts to a common base year. Returns two columns, commodity_year and PPIPR, where PPIPR equals the year's price-received index divided by the index in current_year (so PPIPR(current_year) == 1).

Usage

```
get_price_indices(current_year = NULL)
```

Arguments

current_year	Integer scalar. The base year used for normalization. The returned PPIPR equals 1 for this year.

Details

Data sources (from rfcipDemand):

- nassSurveyPriceRecivedIndex (annual; expects commodity_year, index_for_price_recived).
- nassAgPriceMonthlyIndex (monthly U.S. agricultural price index; expects year, comm, index).

Synthesizing the current year (if missing in the annual table):

- Compute the arithmetic mean of the monthly index where comm == "Agricultural" for both current_year and current_year - 1.
- Multiply last year's annual index_for_price_recived by the ratio mean_monthly(current_year) / mean_monthly(current_year - 1) to derive the current-year annual index.
- Append this row with data_source = "calculated".

Normalization:

- Let the denominator be the (mean) index_for_price_recived among rows with commodity_year == current_year (provides stability if duplicates exist).
- Define PPIPR = index_for_price_recived / denominator.

Output shape:

- Returns only commodity_year and PPIPR, sorted ascending by commodity_year.
- If the input annual table contains multiple rows per year, duplicates are preserved in the output (each with its own PPIPR). Aggregate if you require strictly one row per year (see Notes).

Value

A data.table with two columns:

- commodity_year - integer year.
- PPIPR - numeric deflator equal to the year's price-received index divided by the current_year index.

Assumptions & Notes

- Assumes both reference datasets from **rfcipDemand** are available with the specified columns (including the source's spelling `index_for_price_recived`).
- Monthly means are computed with `na.rm = TRUE`.
- If you need one row per year, post-aggregate: `dt[, .(PPIPR = mean(PPIPR, na.rm = TRUE)), by = commodity_year]`.

`plot_fcip_main_outcomes`

Plot FCIP Main Outcomes (Faceted Bar Charts)

Description

The function expects `data` to contain:

- a numeric/factor year column (specified via `colume_year`);
- a numeric `value` column (bar heights);
- a categorical "outcome" column (specified via `colume_outcome`) used for faceting; and
- an optional grouping column (specified via `colume_grouping`) used to color/ fill bars and determine legend entries.

For legend ordering, the function ranks groups using the subset of rows from the most recent year *within the facet whose label equals* `"(B) Liability amount in U.S. dollars"`. If your outcome labels differ, you may want to harmonize them or adjust the code that builds `labs`.

Usage

```
plot_fcip_main_outcomes(
  data,
  colume_outcome,
  colume_year,
  colume_grouping = NULL,
  time_scale_theme = NULL,
  general_theme = NULL,
  palette = c("#003524", "#00583D", "#A0BD78", "#BED73B", "#D7E5C8", "#FEF389",
    "#FFC425", "#BE5E27", "#9DD9F7", "#51ABA0", "#0F374B")
)
```

Arguments

`data`	A `data.frame` (or `data.table`) with at least the columns: `value`, the column named by `colume_year`, the column named by `colume_outcome`, and (optionally) the column named by `colume_grouping`.
`colume_outcome`	Character scalar. Name of the column in `data` that defines facet panels (e.g., outcome labels such as `"(A) Net reported acres"` and `"(B) Liability amount in U.S. dollars"`).
`colume_year`	Character scalar. Name of the year column in `data` (x-axis, typically the FCIP commodity year).

`colume_grouping`
: Character scalar or `NULL`. Name of the grouping column in `data` used for fill/color and legend entries. If `NULL`, all bars are treated as a single group.

`time_scale_theme`
: Optional `ggplot2` scale/theme element controlling the x-axis (e.g., `scale_x_continuous(...)`). If `NULL`, a default scale is applied using the unique values of the supplied year column.

`general_theme`
: Optional `ggplot2` theme applied to the figure. If `NULL`, `ers_theme()` is used and further tweaked (text sizes, legend layout, facet strips). Requires that `ers_theme()` is available in scope.

`palette`
: Optional character vector of HEX colors used by `scale_fill_manual()`. If provided, it is applied to the grouping legend.

Details

Creates a faceted bar chart of FCIP outcomes over time by a chosen grouping, with a shared x-axis (commodity year) and a legend. Facets are determined by an "outcome" column in `data` (e.g., liability vs. acres), allowing one or more panels depending on the unique values present.

Internally, the function:

1. Copies the columns named by `colume_outcome`, `colume_grouping`, and `colume_year` into standard placeholders used in aesthetics.
2. Builds a ranking of groups from the latest year within the facet whose label equals `"(B) Liability amount in U.S. dollars"`, then maps that ranking to legend order.
3. Draws a bar chart of `value ~ year`, colored/filled by group, and `facet_wrap()`s by outcome (free y-scales).
4. Applies the supplied or default x-axis scale and theme, plus optional manual fill palette.

Value

A `ggplot` object.

Note

The parameter names use `colume_*` (typo preserved for backward compatibility). Consider aliasing/renaming in a future version.

`plot_liability_and_acres` *Plot Liability and Net Reported Acres Faceted by a Grouping Variable*

Description

Produces a two-panel bar chart:

1. **Panel A** - Liability (in U.S.\ dollars) by year and group.
2. **Panel B** - Net reported acres (in acres) by year and group.

The panels share the same x-axis (commodity years) and a single legend that is displayed beneath the figure.

Usage

```
plot_liability_and_acres(
  data,
  grouping_variable,
  grouping_name = NULL,
  time_scale_theme = NULL,
  general_theme = NULL,
  label_liability = "Billion U.S. dollars\n",
  label_net_reported_acres = "Million acres\n",
  palette = c("#003524", "#00583D", "#A0BD78", "#BED73B", "#D7E5C8", "#FEF389",
    "#FFC425", "#BE5E27", "#9DD9F7", "#51ABA0", "#0F374B")
)
```

Arguments

`data`	A `data.frame` containing at least these columns: • `commodity_year` - numeric or factor; the x-axis. • `value` - numeric; the bar heights. • `variable` - factor with the levels `"(A) Liability in U.S. dollars"` and `"(B) Net reported acres"`. • The column referenced by `grouping_variable`.
`grouping_variable`	A character scalar giving the name of the column in `data` that defines the groups (e.g.\ `"commodity"`, `"state"`).
`grouping_name`	Optional character scalar used as the legend title. Defaults to the value of `grouping_variable` if `NULL`.
`time_scale_theme`	Optional `ggplot2` scale or theme element that controls the x-axis breaks/labels. If `NULL`, the function applies `scale_x_continuous(breaks = unique(data$commodity_year), labels = unique(data$commodity_year))`.
`general_theme`	Optional `ggplot2` theme applied to both panels. If `NULL`, `ers_theme()` is used with additional tweaks for font sizes, legend layout, and facet strips.
`label_liability`	Y-axis label for Panel A. Default `"Billion U.S. dollars\\n"`.
`label_net_reported_acres`	Y-axis label for Panel B. Default `"Million acres\\n"`.
`palette`	A character vector of hex colors used to fill the bars. The default is an 11-color palette aligned with ERS/NDSU branding.

Details

Internally the function:

1. Duplicates the column named by `grouping_variable` into `data$grouping_variable` for convenient aesthetic mapping.
2. Builds two separate `ggplot` bar charts (one per `variable` level), applies user themes and palettes, and hides their legends.
3. Extracts a shared legend and x-axis grob with `gtable_filter()`.
4. Assembles everything with `gridExtra::grid.arrange()`.

Value

A named list with five objects:

`fig` A `gtable` containing the assembled two-panel figure, shared x-axis label, and bottom legend.

`figA` The `ggplot` object for Panel A.

`figB` The `ggplot` object for Panel B.

`ldgnd` The extracted legend as a `gtable`.

`xlabT` The shared x-axis grob as a `gtable`.

Examples

```
## Not run:
# Group by commodity with a custom x-axis theme
plot_liability_and_acres(
  data = data_comm,
  grouping_variable = "commodity",
  grouping_name = "Commodity",
  time_scale_theme = ggplot2::scale_x_continuous(
    breaks = seq(2008, 2024, 2),
    labels = seq(2008, 2024, 2)
  ),
  general_theme = ggplot2::theme_minimal()
)

## End(Not run)
```

---

plot_prf_statistics *Plot PRF Summary Statistics on the Official Grid*

Description

Creates a choropleth map of Pasture, Rangeland, Forage (PRF) statistics on the official PRF grid. Optionally facets the map by a disaggregate variable (e.g., data source: Official vs. Study Approximation).

Usage

```
plot_prf_statistics(
  data,
  outcome_variable,
  plot_title,
  prf_polygon = NULL,
  disaggregate_variable = NULL,
  palette = c("#FEF9C5", "#FFC425", "#FEF389", "#E7F2B4", "#D7E5C8", "#C5DE91",
    "#BED73B", "#A0BD78", "#00583D", "#0F374B")
)
```

Arguments

data	A data frame containing at least grid_id and the variable specified in outcome_variable. If disaggregate_variable is used, it must also be present in data.
outcome_variable	Character scalar giving the name of the variable in data to map (e.g., a binned category such as "mean_cat").
plot_title	Character scalar used as the main plot title.
prf_polygon	Optional sf object containing PRF grid polygons with a grid_id column. If NULL, the function calls get_official_prf_polygon().
disaggregate_variable	Optional character scalar giving the name of a column in data used for faceting (e.g., "disaggregate" or "source"). If NULL, no facets are drawn.
palette	Character vector of hex color codes used for the fill scale. The default is a 10-color sequential/diverging palette tailored to PRF maps.

Details

The function:

1. Loads the official PRF grid polygons via get_official_prf_polygon() if prf_polygon is not supplied.
2. Filters PRF polygons to those with grid_id present in data.
3. Transforms the PRF polygons to match the CRS of the state boundaries from urbnmapr::get_urbn_map(map = "states").
4. Joins data to the PRF polygons by grid_id.
5. Filters out rows with missing values in outcome_variable.
6. Plots a choropleth map using the supplied palette, overlaying U.S. state boundaries (excluding Alaska and Hawaii; FIPS 2 and 15).
7. Optionally facets the plot by disaggregate_variable, with two columns of panels.

The function assumes that data contains a column named grid_id matching the PRF grid IDs, and a column whose name is given by outcome_variable. When disaggregate_variable is provided, it should also be a column in data.

Value

A ggplot object representing the PRF statistics map, optionally with facets by disaggregate_variable.

Examples

```
## Not run:
p <- plot_prf_statistics(
  data               = df,
  outcome_variable   = "mean_cat",
  plot_title         = "PRF Index Mean (2016-2024)",
  disaggregate_variable = "disaggregate"
)
print(p)

## End(Not run)
```

`plot_us_states_choropleth` *Plot an Annotated US States Choropleth Map*

Description

Creates a choropleth map of US states colored by a categorical value, with state labels and special treatment for small states (VT, NH, and other small ones) using repelled text. Optionally embeds a table grob in the bottom-left corner of the map.

Usage

```
plot_us_states_choropleth(
  data,
  legend_title = NULL,
  palette = c("#BE5E27", "#FFC425", "#FEF389", "#BED73B", "#A0BD78", "#00583D",
    "#003524", "#D7E5C8", "#9DD9F7", "#51ABA0", "#0F374B"),
  table_grob = NULL,
  label_size = 2.5,
  na.value = "white",
  keep_all_states = FALSE
)
```

Arguments

`data`	A data frame containing at least: • `state_code`: numeric FIPS code matching `urbnmapr::get_urbn_map("states")` • `value_cat`: categorical variable used for fill colors • `value`: numeric variable used to display numeric labels
`legend_title`	Character; title for the fill legend. If `NULL`, no title is shown.
`palette`	Character vector of colours (hex codes) to use for the categories. Defaults to a 10-colour NDSU-inspired palette.
`table_grob`	A `grob` object (e.g. from `gridExtra::tableGrob`) to annotate on the map; if `NULL`, no table is added.
`label_size`	label size for sgeom_sf_text
`na.value`	The aesthetic value to use for missing (NA) values
`keep_all_states`	keep all states

Details

- Uses `urbnmapr::get_urbn_map(map = "states", sf = TRUE)` to fetch a US states basemap.
- Joins the input data on `state_code` and filters out states with missing `value_cat`.
- Computes equal-area centroids (EPSG:5070) to place labels.
- Flags states with area < 50,000 km^2 as "small" and applies repelled text labels.
- Standard states get text labels placed via `geom_sf_text()`.
- Small states in the east and west are nudged horizontally; VT & NH get custom nudges.
- Adds an optional table in the bottom-left via `annotation_custom()`.

Value

A ggplot object showing the US states choropleth with annotated labels.

`policytime` *Add U.S. Farm Policy Vertical Lines and Labels to a ggplot*

Description

This function overlays vertical lines and text labels on a ggplot object to mark major U.S. agricultural policy events (e.g., Farm Bills or Acts). The vertical lines are drawn at specific years, and labels are positioned based on values provided in the `pty` vector.

Usage

```
policytime(pty, plot, size)
```

Arguments

`pty`	A numeric vector of y-axis positions for label placement. The length of the vector determines which policy annotations are added: • 1st: 1980 Act • 2nd: 1994 Act • 3rd: 1996 Farm Bill • 4th: 2000 Agricultural Risk Protection Act • 5th: 2008 Farm Bill • 6th: 2014 Farm Bill • 7th: 2018 Farm Bill
`plot`	A ggplot object to which policy lines and labels will be added.
`size`	A numeric value indicating the text size of the policy labels.

Details

The lines and labels are added in brown color with dashed lines (`lty=5`), and labels are rotated vertically. Labels are only drawn if the corresponding index exists in `pty`.

Value

A ggplot object with added vertical dashed lines and corresponding text labels for each policy year provided.

Examples

```
## Not run:
base_plot <- ggplot(data, aes(x = commodity_year, y = value)) +
  geom_line()
policy_positions <- c(10, 15, 20, 25, 30, 35, 40)
annotated_plot <- policytime(policy_positions, base_plot, size = 3)
print(annotated_plot)

## End(Not run)
```

`reshape_statistics` *Reshape and Label Balance Statistics*

Description

Reshapes selected balance statistics from a data frame into long format and assigns descriptive labels to each statistic disaggregate. This function is commonly used to prepare datasets for balance comparison plots or summary tables.

Usage

```
reshape_statistics(
  data,
  statistic_variables,
  disaggregate_labels,
  rename = NULL
)
```

Arguments

`data`	A data frame containing `grid_id` and the statistic variables to be reshaped.
`statistic_variables`	Character vector of column names in `data` that contain the statistics to reshape.
`disaggregate_labels`	Character vector of descriptive labels corresponding to `statistic_variables`. Must be the same length and in the same order.
`rename`	Optional character scalar. If supplied, the output column `value` will be renamed to this string (e.g., `"std_diff"` or `"index"`).

Details

The function:

1. Validates that `statistic_variables` and `disaggregate_labels` have equal length.
2. Checks that all specified statistic variables exist in `data`.
3. Selects `grid_id` and the variables listed in `statistic_variables`.
4. Converts the data from wide to long format using `tidyr::gather()`, creating columns: `grid_id`, `disaggregate`, and `value`.
5. Converts `disaggregate` to a factor and replaces raw column names with human-readable labels using `disaggregate_labels`.
6. Optionally renames the `value` column using the `rename` argument.

Value

A long-format data frame with the columns:

grid_id Identifier for the spatial/grid unit.

disaggregate Factor indicating the origin of each statistic, labeled using `disaggregate_labels`.

value Numeric value of the statistic, or renamed to `rename` if provided.

setup_environment *Setup Project Environment*

Description

Initializes the working environment for a project by creating required directories, setting useful global options, and fixing the random seed.

Usage

```
setup_environment(
  year_beg = 2001,
  year_end = as.numeric(format(Sys.Date(), "%Y")),
  seed = 1980632,
  project_name,
 local_directories = list(file.path("data-raw", "output"), file.path("data-raw",
    "scripts"), file.path("data")),
  fastscratch_root = NULL,
  fastscratch_directories = NULL
)
```

Arguments

year_beg	Integer. Beginning year of the analysis (default: 2001).
year_end	Integer. Ending year of the analysis (default: current system year).
seed	Integer. Random seed for reproducibility (default: 1980632).
project_name	Character. Project name (required). Used to build fast-scratch directory paths.
local_directories	List of project-local directories to create (default: `list("data-raw/output", "data-raw/scripts", "data")`).
fastscratch_root	Optional character. Root directory for fast-scratch files. If `NULL`, it is set automatically: • Windows: `"C:/fastscratch"` • Linux/macOS: `"/fastscratch/<username>"`
fastscratch_directories	List of fast-scratch subdirectories (relative to `<fastscratch_root>/<project_name>`) to create. If `NULL`, no fast-scratch subdirectories are created and `wd` is returned as an empty list.

Details

The function ensures the requested directories exist, creating them if necessary. Directory keys in the returned `wd` list are the basenames of the provided `fastscratch_directories`.

It also sets the following options:

- `options(scipen = 999)` (turns off scientific notation)
- `options(future.globals.maxSize = 8 * 1024^3)` (~8 GiB)
- `options(dplyr.summarise.inform = FALSE)` (quiet **dplyr**)

Finally, the random number generator is seeded with the provided `seed`.

Value

A list with:

wd Named list of created fast-scratch directories. Empty if `fastscratch_directories = NULL`.

year_beg Starting year (integer).

year_end Ending year (integer).

seed Seed value used for RNG.

Index