Package: rpspanel (via r-universe)

October 28, 2026

Type Package

Title US State Energy-Economy Panels with Renewable Portfolio Standards

Version 0.1.0.9000

Description Builds a documented, reproducible state-by-year panel for the 50 US states and the District of Columbia that combines energy consumption, prices and production from the US Energy Information Administration (EIA) State Energy Data System, electricity generation and retail sales, state energy-related carbon dioxide emissions, and state GDP, income and population from the Bureau of Economic Analysis (BEA). Data are downloaded on demand through the official APIs and cached locally, with the source, vintage and transformation of every variable recorded in a codebook. Renewable Portfolio Standard (RPS) rules are turned into standardised annual stringency indices (adoption, nominal, coverage-adjusted and incremental requirements, following Yin and Powers (2010) <doi:10.1016/j.enpol.2009.10.067>, plus design-flexibility indicators). Output works directly with the 'plm', 'fixest' and 'csdm' packages.

License MIT + file LICENSE

URL https://github.com/adebayomoses/rpspanel, https://adebayomoses.github.io/rpspanel/

BugReports https://github.com/adebayomoses/rpspanel/issues

Depends R (>= 4.1.0)

Imports cli, httr2 (>= 1.0.0), stats, tools, utils

Suggests arrow, fixest, haven, knitr, plm, rmarkdown, testthat (>= 3.1.7), withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libssl-dev

Repository https://ropensci-staging.r-universe.dev

Date/Publication 2026-10-28 12:19:10 UTC

RemoteUrl https://github.com/adebayomoses/rpspanel

RemoteRef HEAD

RemoteSha e12067a2a98eff5b55c6d5260b82b14ad120f4d0

Contents

as_pdata 2
build_panel 3
cache 5
cite_sources 5
codebook 6
convert_units 7
deflate 7
export_panel 8
fetch_bea 9
fetch_census 10
fetch_eia 11
fetch_rps 12
harmonise_states 13
rps_rules 13
rps_stringency 14
us_states 16

Index **18**

`as_pdata` *Convert a panel for estimation packages*

Description

Prepares a panel for `plm`, `fixest` or `csdm`. The package does not re-implement estimators; it hands a correctly indexed panel to them.

Usage

```
as_pdata(x, to = c("plm", "fixest", "csdm", "data.frame"))
```

Arguments

x	An rps_panel (or any data frame with state and year).
to	Target package.

Details

- "plm": a pdata.frame indexed by state and year.
- "fixest": a fixest_panel (for lag and lead operators) indexed by state and year.
- "csdm": a plain data frame sorted by state and year, with state as a factor; pass id = "state" and time = "year" to csdm.
- "data.frame": the same sorted plain data frame.

Value

An object of the requested class.

Examples

```
p <- build_panel(states = c("CA", "TX"), years = 2005:2020, groups = "rps",
                 rps_methods = c("adoption", "nominal"))
head(as_pdata(p, "data.frame"))
if (requireNamespace("plm", quietly = TRUE)) {
  class(as_pdata(p, "plm"))
}
```

---

build_panel *Build a state-year energy-economy panel*

Description

The main entry point of the package. Downloads (or reads from the cache) the requested EIA and BEA series, harmonises state codes and units, merges them into one tidy state-by-year table, adds derived variables and RPS stringency measures, and records the source and vintage of every variable.

Usage

```
build_panel(
  states = NULL,
  years = 2001:2022,
  vars = NULL,
  groups = panel_groups,
  rps_methods = c("adoption", "nominal", "coverage_adjusted", "incremental",
    "flexibility"),
  rps_rules = NULL,
  price_base_year = 2017,
```

```
  eia_key = NULL,
  bea_key = NULL,
  cache = TRUE,
  refresh = FALSE,
  quiet = FALSE,
  ...
)
```

Arguments

states	States to include (any form accepted by harmonise_states()); NULL for all 50 states and DC.
years	Years to include.
vars	Variable names to include (see codebook()). Derived variables pull in the series they need. Overrides groups.
groups	Variable groups to include when vars is NULL: any of "energy", "electricity", "emissions", "economy", "rps".
rps_methods	Methods passed to rps_stringency().
rps_rules	An rps_rules object; defaults to the bundled coding.
price_base_year	Base year for real prices.
eia_key, bea_key	API keys; default to the EIA_API_KEY and BEA_API_KEY environment variables.
cache, refresh	Use the download cache / force a fresh download.
quiet	Suppress progress messages.
...	Further arguments to rps_stringency(), such as interpolation.

Details

If one source cannot be reached (no API key, no internet, service down) its variables are left out with a warning and the rest of the panel is still returned.

Value

A data frame of class rps_panel with one row per state and year, columns state, state_name, fips, year and the requested variables. Attributes: sources (source, vintage, URL), codebook (definitions of the included variables) and call.

Examples

```
# RPS measures only: no API key needed
p <- build_panel(states = c("CA", "TX", "NY"), years = 2000:2025,
                 groups = "rps",
                 rps_methods = c("adoption", "nominal", "flexibility"))
head(p)
```

```
p <- build_panel(years = 2005:2022)
codebook(p)
```

---

cache *Manage the download cache*

Description

Downloads are cached as `.rds` files so that a panel can be rebuilt offline and so that a data vintage stays pinned until you refresh it. `rps_cache_dir()` returns the cache directory, `rps_cache_list()` lists cached downloads and `rps_cache_clear()` deletes them.

Usage

```
rps_cache_dir()

rps_cache_list()

rps_cache_clear()
```

Details

The cache directory defaults to `tools::R_user_dir("rpspanel", "cache")` and can be changed with `options(rpspanel.cache_dir = "path")`.

Value

`rps_cache_dir()`: a path. `rps_cache_list()`: a data frame with `file`, `source`, `retrieved` and `size_kb`. `rps_cache_clear()`: the number of files removed, invisibly.

Examples

```
rps_cache_dir()
```

---

cite_sources *Citations for the data sources used*

Description

Lists the citation for every source used in a panel, including the retrieval date (vintage), plus the citation for the package itself.

Usage

```
cite_sources(x = NULL, format = c("text", "bibtex"))
```

Arguments

x	An rps_panel, a character vector of source keys (see source in codebook()), or NULL for all sources.
format	"text" for plain-text references or "bibtex".

Value

A character vector of references of class rps_citation, which prints one reference per paragraph.

Examples

```
cite_sources(c("eia_seds", "bea_regional"))
p <- build_panel(states = "CA", years = 2010:2020, groups = "rps",
                 rps_methods = "nominal")
cite_sources(p, format = "bibtex")
```

---

codebook *Variable codebook*

Description

Returns the definition, unit, source, source series identifier and transformation of every variable the package can produce, or of the variables in a panel built with build_panel().

Usage

```
codebook(x = NULL, group = NULL)
```

Arguments

x	NULL (all variables), a character vector of variable names, or an rps_panel.
group	Optional group filter: "energy", "electricity", "emissions", "economy" or "rps".

Value

A data frame with columns variable, group, label, unit, source, series and transformation.

Examples

```
codebook(group = "rps")
codebook(c("co2_pc", "gdp_real"))
```

convert_units — *Convert between units*

Description

Converts energy, mass, money-scale and population units used by EIA and BEA. Electricity is converted to Btu at the thermal equivalent of 3,412.14 Btu per kWh (the convention used in SEDS for electricity consumption).

Usage

```
convert_units(x, from, to)

convert_units_available()
```

Arguments

x	Numeric vector.
from, to	Unit names (case-insensitive), for example "billion Btu", "MMBtu", "thousand MWh", "million metric tons", "short ton", "million dollars". See convert_units_available().

Value

Numeric vector in the to unit.

Examples

```
convert_units(1, "trillion Btu", "billion Btu")
convert_units(1000, "thousand MWh", "TWh")
convert_units(1, "million metric tons", "short ton")
```

deflate — *Convert nominal values to real values*

Description

Divides by a price index and rescales so that values are expressed in prices of base_year.

Usage

```
deflate(
  x,
  price_index,
  years = NULL,
  base_year = NULL,
  index_years = years,
  index_values = price_index
)
```

Arguments

`x`	Numeric vector of nominal values.
`price_index`	Price index for the same observations as `x` (for example the BEA GDP deflator, 2017 = 100).
`years`	Optional years of the observations; needed with `base_year`.
`base_year`	Year whose prices the result is expressed in. If `NULL`, the index is assumed to equal 100 in the base year and the result is `x * 100 / price_index`.
`index_years`, `index_values`	Optional full index series used to look up the base-year value when `base_year` is not among `years`.

Value

Numeric vector of real values.

Examples

```
# $10 in 2010 and $12 in 2020 with deflator 90 and 110 (2017 = 100)
deflate(c(10, 12), c(90, 110))
# Expressed in 2020 prices instead
deflate(c(10, 12), c(90, 110), years = c(2010, 2020), base_year = 2020)
```

`export_panel` *Export a panel to CSV, Stata or Parquet*

Description

Writes the panel and, by default, its codebook next to it (`<name>_codebook.csv`). Stata files carry variable labels from the codebook.

Usage

```
export_panel(x, path, format = NULL, codebook = TRUE)
```

Arguments

`x`	An `rps_panel`.
`path`	Output file path.
`format`	`"csv"`, `"stata"` or `"parquet"`. Guessed from the file extension (`.csv`, `.dta`, `.parquet`) when `NULL`.
`codebook`	Also write the codebook as CSV.

Value

The path, invisibly.

Examples

```
p <- build_panel(states = "CA", years = 2010:2020, groups = "rps",
                 rps_methods = "nominal")
f <- tempfile(fileext = ".csv")
export_panel(p, f)
head(read.csv(f))
```

---

fetch_bea *Download data from the BEA API*

Description

Retrieves state series from the Bureau of Economic Analysis Regional accounts, or national series from the NIPA tables (used for price deflators).

Usage

```
fetch_bea(
  table,
  line_code,
  dataset = c("Regional", "NIPA"),
  states = NULL,
  years = NULL,
  api_key = NULL,
  cache = TRUE,
  refresh = FALSE
)
```

Arguments

table	BEA table name, for example "SAGDP9" or "T10109".
line_code	Line code (Regional) or line number (NIPA).
dataset	"Regional" (state data) or "NIPA" (national data).
states	States to keep (Regional only); NULL keeps all 50 + DC.
years	Years to keep; NULL keeps all available years.
api_key	BEA API key. Defaults to the BEA_API_KEY environment variable.
cache	Use the local download cache.
refresh	Ignore any cached copy and download again.

Details

Useful Regional tables and line codes:

- SAGDP9 line 1: real GDP, all industries (millions of chained 2017 dollars); line 12: manufacturing.

- SAGDP2 line 1: nominal GDP (millions of current dollars).
- SAINC1 line 1: personal income (millions of dollars); line 2: population (persons); line 3: per capita personal income (dollars).

Useful NIPA table: T10109 line 1, the implicit price deflator for GDP (index, 2017 = 100).

Value

A data frame with columns state (NA for NIPA data), year, series ("<table>:<line>"), value and unit. Values are in the units reported by BEA (the UNIT_MULT scale factor is already applied to the unit label, not the value).

Examples

```
fetch_bea("SAGDP9", 1, states = "CA", years = 2015:2020)
fetch_bea("T10109", 1, dataset = "NIPA")
```

---

fetch_census *Download state data from the Census Bureau API*

Description

Retrieves a state-level variable from the American Community Survey 1-year estimates (default: total population, B01003_001E). This is an optional supplement: build_panel() takes population from BEA, which is available for every year. ACS 1-year estimates start in 2005, and the standard 2020 release does not exist, so that year is skipped.

Usage

```
fetch_census(
  years,
  variable = "B01003_001E",
  states = NULL,
  api_key = NULL,
  cache = TRUE,
  refresh = FALSE
)
```

Arguments

years	Years to download (2005 onward).
variable	ACS variable code.
states	States to download; any form accepted by harmonise_states(). NULL means all 50 states and DC.
api_key	Census API key. Defaults to the CENSUS_API_KEY environment variable. The Census API rejects requests without a key; register for a free one at https://api.census.gov/data/key_signup.html.

cache	Use the local download cache.
refresh	Ignore any cached copy and download again.

Value

A data frame with columns `state`, `year`, `series`, `value` and `unit` ("persons" for the default population variable, otherwise `NA`; look up other ACS variables in the Census documentation). Years that fail to download are skipped with a warning.

Examples

```
fetch_census(2019, states = c("CA", "TX"))
```

---

fetch_eia *Download state data from the EIA API*

Description

Queries version 2 of the US Energy Information Administration API and returns tidy long data. Results are cached (see `rps_cache_dir()`); the API key is never written to the cache.

Usage

```
fetch_eia(
  dataset = c("seds", "co2", "generation", "retail_sales"),
  series,
  states = NULL,
  years = NULL,
  api_key = NULL,
  cache = TRUE,
  refresh = FALSE
)
```

Arguments

dataset	One of `"seds"`, `"co2"`, `"generation"`, `"retail_sales"`.
series	Character vector of series identifiers (see Details).
states	States to download; any form accepted by `harmonise_states()`. `NULL` means all 50 states and DC.
years	Integer vector of years. `NULL` downloads all available years.
api_key	EIA API key. Defaults to the `EIA_API_KEY` environment variable.
cache	Use the local download cache.
refresh	Ignore any cached copy and download again.

Details

Supported datasets:

- "seds": State Energy Data System. series are SEDS MSN codes, for example "TETCB" (total energy consumption, billion Btu).
- "co2": energy-related CO2 emissions aggregates for all fuels. series are EIA sector ids: "TT" (total), "EC" (electric power), "IC" (industrial), "TC" (transportation), "RC" (residential), "CC" (commercial).
- "generation": net generation, all sectors (forms EIA-923/906/920). series are fuel type ids such as "ALL", "COW", "NG", "NUC", "HYC", "WND", "SUN", "GEO", "WWW", "WAS".
- "retail_sales": electricity retail sales and average price, all sectors. series are "sales" and/or "price".

Value

A data frame with columns state, year, series, value and unit, with attributes rps_source and rps_retrieved (download time, UTC).

Examples

```
fetch_eia("seds", "TETCB", states = c("CA", "TX"), years = 2015:2020)
```

---

fetch_rps — *Get RPS rule coding from the package or a URL*

Description

Without arguments, returns the bundled coding (same as rps_rules()). With URLs, downloads CSV files that follow the rps_rules() layout, for example a shared or corrected version of the coding, and caches them. The package does not redistribute DSIRE, LBNL or PUDL data; if you build your own coding from those sources, check their terms of use.

Usage

```
fetch_rps(rules_url = NULL, targets_url = NULL, cache = TRUE, refresh = FALSE)
```

Arguments

rules_url, targets_url	URLs (or local paths) of CSV files.
cache	Use the local download cache.
refresh	Ignore any cached copy and download again.

Value

An rps_rules object.

Examples

```
fetch_rps()
```

---

harmonise_states *Harmonise state identifiers*

Description

Converts state names, USPS abbreviations or FIPS codes (two-digit, numeric, or five-digit BEA `GeoFips` such as `"06000"`) to a single representation. Matching ignores case, surrounding white space and trailing footnote markers such as the asterisks BEA appends to some state names.

Usage

```
harmonise_states(x, to = c("abbr", "name", "fips"), warn = TRUE)
```

Arguments

`x`	A character or numeric vector of state identifiers.
`to`	Output representation: `"abbr"` (default), `"name"` or `"fips"`.
`warn`	If `TRUE`, warn about values that could not be matched (for example `"US"` or regional aggregates). Unmatched values become `NA`.

Value

A character vector the same length as `x`.

Examples

```
harmonise_states(c("California", "tx", "06", 36, "11000", "Alaska *"))
harmonise_states(c("CA", "NY"), to = "fips")
```

---

rps_rules *Coded Renewable Portfolio Standard rules*

Description

Reads the RPS rule coding that feeds `rps_stringency()`. By default this is the coding bundled with the package in `inst/extdata/`; you can point to your own files with the same layout to correct or extend it.

Usage

```
rps_rules(states = NULL, rules_path = NULL, targets_path = NULL)
```

Arguments

`states` Optional states to keep.

`rules_path, targets_path` Paths to CSV files with your own coding. Default: the files bundled with the package.

Details

The coding has two tables:

rules (one row per state): `has_rps` (1 = mandatory RPS, 0 = none), `enacted_year`, `first_compliance_year`, `repeal_year`, `target_type` (`"share"` of retail sales, or `"capacity_mw"`), `final_target`, `final_target_year`, `coverage_share` (share of retail sales covered), `out_of_state_rec`, `carveout`, `acp` (0/1 design indicators), `carveout_types`, `acp_usd_mwh`, `baseline_re_share` (optional existing renewable share at enactment; filled from data by `build_panel()`), `source`, `verified` and `notes`. Optional: `baseline_includes_hydro` (1 if the targets count existing hydro, so the incremental baseline should too; default 0).

targets (one row per legal target): `state`, `year`, `target` (share of sales between 0 and 1, or MW for capacity targets).

States that do not appear in the rules table are treated as *not yet coded*, and their stringency measures are `NA` rather than zero.

Value

An object of class `rps_rules`: a list with data frames `rules` and `targets`.

Status of the bundled coding

The bundled coding covers all 50 states and DC. Rows with `verified = TRUE` have been hand-checked against the statute, the DSIRE program page and the LBNL RPS & CES data, as set out in the package's validation plan; `source` and `notes` give the citations and coding choices. Please report corrections on the issue tracker.

Examples

```
r <- rps_rules()
r
head(r$targets)
```

---

rps_stringency *Annual RPS stringency indices*

Description

Turns coded RPS rules (`rps_rules()`) into comparable state-year measures, so you can study how strong a policy is, not only whether it exists.

Usage

```
rps_stringency(
  years,
  states = NULL,
 method = c("adoption", "nominal", "coverage_adjusted", "incremental", "flexibility"),
  rules = rps_rules(),
  baseline = NULL,
  interpolation = c("linear", "step"),
  adopt_from = c("enactment", "first_compliance"),
  floor_zero = FALSE,
  sales = NULL,
  capacity_factor = 0.33
)
```

Arguments

`years`	Years to compute.
`states`	States to include (default: all 50 + DC).
`method`	One or more of `"adoption"`, `"nominal"`, `"coverage_adjusted"`, `"incremental"`, `"flexibility"`.
`rules`	An `rps_rules` object; defaults to the bundled coding.
`baseline`	Optional data frame with columns `state` and `baseline_share`, the existing (eligible) renewable share of sales or generation at enactment. Overrides `baseline_re_share` in the rules.
`interpolation`	`"linear"` or `"step"`.
`adopt_from`	Year from which `adoption` is 1: `"enactment"` or `"first_compliance"`.
`floor_zero`	If `TRUE`, negative incremental requirements are set to 0.
`sales`	Optional data frame with columns `state`, `year` and `sales_gwh` (retail electricity sales, GWh, the same as EIA's million kWh), used to convert capacity-based standards into shares. `build_panel()` supplies it from EIA retail sales.
`capacity_factor`	Assumed capacity factor of the resources that meet a capacity-based standard. The default 0.33 is typical of wind, which met most of the Texas and Iowa requirements.

Details

Methods:

- `adoption`: 1 if an RPS is in force that year (from `adopt_from` until any repeal), else 0.
- `nominal`: required renewable share of retail sales in that year. Zero before the first compliance year. Between legal targets the requirement is interpolated linearly (`interpolation = "linear"`) or held at the last legal target (`"step"`); after the final target year it is held constant. If the first coded target lies after the enactment year, linear interpolation starts from zero at enactment. Capacity-based (MW) standards (Texas, Iowa) are converted to an equivalent share of covered sales: `MW * capacity_factor * 8.76 / (sales_gwh * coverage_share)`, the same approach Berkeley Lab uses. This needs `sales`; without it they are `NA`.

- coverage_adjusted: nominal requirement times the share of retail sales covered by the standard.
- incremental: requirement beyond the renewable share the state already had at enactment, following Yin and Powers (2010): nominal * coverage - baseline_share while the nominal requirement is positive, otherwise 0. The baseline is taken from baseline, or from the baseline_re_share column of the rules. Also returns the baseline used (rps_baseline_share). build_panel() measures the baseline the same way the target is defined: including hydro for states whose targets count existing hydro (baseline_includes_hydro = 1, New York), excluding it otherwise.
- flexibility: indicators for out-of-state renewable credits, carve-outs and alternative compliance payments, set to 0 when no RPS is in force.

States missing from the rule coding get NA for every measure.

Value

A data frame with state, year and the columns for each method: rps_adoption, rps_nominal, rps_coverage_adj, rps_incremental, rps_baseline_share, rps_oos_rec, rps_carveout, rps_acp.

References

Yin, H., & Powers, N. (2010). Do state renewable portfolio standards promote in-state renewable generation? *Energy Policy*, 38(2), 1140-1149.

Examples

```
s <- rps_stringency(2000:2030, states = c("CA", "WA", "TX", "FL"),
                    method = c("adoption", "nominal"))
s[s$state == "WA" & s$year %in% 2011:2021, ]

# Capacity (MW) standards become shares when sales are supplied
tx_sales <- data.frame(state = "TX", year = 2015, sales_gwh = 380000)
rps_stringency(2015, "TX", method = "nominal", sales = tx_sales)

# Step interpolation follows the legal schedule exactly
rps_stringency(2012:2020, states = "WA", method = "nominal",
               interpolation = "step")
```

---

us_states *US states and the District of Columbia*

Description

Returns the 51 geographic units covered by the package with their USPS abbreviation, name and two-digit FIPS code.

Usage

```
us_states()
```

Value

A data frame with columns `abbr`, `name` and `fips`.

Examples

```
head(us_states())
```

Index

as_pdata, 2

build_panel, 3
build_panel(), *6*, *10*, *14–16*

cache, 5
cite_sources, 5
codebook, 6
codebook(), *4*, *6*
convert_units, 7
convert_units_available
(convert_units), 7

deflate, 7

export_panel, 8

fetch_bea, 9
fetch_census, 10
fetch_eia, 11
fetch_rps, 12

harmonise_states, 13
harmonise_states(), *4*, *10, 11*

rps_cache_clear (cache), 5
rps_cache_dir (cache), 5
rps_cache_dir(), *11*
rps_cache_list (cache), 5
rps_rules, 13
rps_rules(), *12*, *14*
rps_stringency, 14
rps_stringency(), *4*, *13*

us_states, 16