Package 'LLMRpanel'

October 29, 2026

Type Package

Title Benchmarked Silicon Samples for Survey and Experiment Design

Version 0.6.3

Description Administers survey and experimental instruments to panels of language-model personas, with option-order randomization and benchmark comparison against human data. Conjoint effects are estimated from recorded respondent-level profile assignments. Samples of language-model personas follow Argyle et al. (2023) <doi:10.1017/pan.2023.2>; the case for benchmarking them against human data is set out in Bisbee et al. (2024) <doi:10.1017/pan.2024.5>; the conjoint estimand is the average marginal component effect of Hainmueller et al. (2014) <doi:10.1093/pan/mpt024>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Depends R (>= 4.2)

Imports LLMR (>= 0.8.9), tibble, rlang, cli, stats, utils

Suggests testthat (>= 3.0.0), ggplot2, knitr, rmarkdown, shiny, bslib, DT, LLMR.shiny (>= 0.1.2)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://github.com/asanaei/LLMRpanel, https://asanaei.github.io/LLMRpanel/

BugReports https://github.com/asanaei/LLMRpanel/issues

NeedsCompilation no

Author Ali Sanaei [aut, cre, cph]

Maintainer Ali Sanaei <sanaei@uchicago.edu>

Repository CRAN

Date/Publication 2026-10-29 00:20:02 UTC

Contents

anes_2024_personas . . . 2
as_persona_frame . . . 5
conjoint_amce . . . 6
conjoint_design . . . 7
conjoint_instrument . . . 8
panel_administer . . . 9
panel_batch_fetch . . . 11
panel_batch_status . . . 12
panel_batch_submit . . . 13
panel_benchmark . . . 14
panel_bias_audit . . . 15
panel_from_data . . . 16
panel_from_margins . . . 18
panel_from_personas . . . 19
panel_generics . . . 21
panel_instrument . . . 22
panel_items . . . 23
panel_usage . . . 24
plot.panel_responses . . . 25
run_panel_studio . . . 26

Index **27**

`anes_2024_personas` *Persona profiles of 100 respondents to the ANES 2024 Time Series Study*

Description

A data frame describing 100 respondents to the American National Election Studies (ANES) 2024 Time Series Study by their demographics and their political attitudes, written as text a language model can read. It serves as ready-made persona material: `panel_from_personas()` turns its rows into a panel of language-model personas. FocusGroup ships an identical copy.

Usage

```
anes_2024_personas
```

Format

A data frame with 100 rows and 123 columns. Each row is one respondent; no ANES case identifier is kept. Column names are handles whose prefix names a block, so `startsWith(names(x), "att_iss_")` selects the issue positions:

- demo_: 17 demographic fields: age band, sex, education, race/ethnicity, marital status, household income, religion and its importance, religious attendance, census region, rural-urban self-identification, employment status, home ownership, children in the household, union household, military service, and attention to politics.
- att_id_: 9 fields of party identification, ideology, and vote choice.
- att_aff_: 18 feeling thermometers toward candidates, parties, institutions, and groups.
- att_eval_: 14 evaluations: approval of the President, Congress, and the Supreme Court, the economy, personal finances, feelings about the country, and which party would better handle the economy and immigration.
- att_iss_: 47 issue positions, among them taxes and spending, immigration, guns, climate, health care, abortion, policing, race, and foreign policy.
- att_val_: 17 fields of trust, values, and views of democracy.
- ideology_score: A number placing the respondent on one liberal-conservative dimension (see Details); the rows are sorted by it.

Every column but ideology_score holds character values:

- An answer with a value label is written as that label without its code ("Strong Democrat", "Favor a great deal"). Parenthetical qualifiers are kept, the interviewer notes "{VOL}" and "{SPECIFY}" are removed, and the four labels of the immigration policy item (V241386) that the Stata file cuts at 80 characters are restored in full from the codebook.
- The six 7-point scales that ANES labels only at their ends are written as sentences, endpoints included: "5 on a scale from 1 (Government should provide many fewer services) to 7 (Government should provide many more services)".
- Feeling thermometers are written as "70 on a 0-100 feeling thermometer".
- "Haven't thought much about this" is kept as an answer where ANES offered it.
- Age is given in six bands and children in the household as "Yes" or "No". Employment status is written from the full ANES labels without their "R " prefix ("Retired (if also working, working < 20 hrs/wk)"). Religious attendance is "Never" for respondents who said they never attend services (V241439 = 2), who were not asked how often (V241440).
- NA marks a missing-data code: refused, don't know, inapplicable, no post-election interview, and so on, and a thermometer rating outside 0-100 ("don't know where to rate", "don't recognize"). The 100 rows hold 246 such cells, most of them in the vote items, which do not apply to everyone.

The data frame carries these attributes:

- dictionary: A data frame with one row per column except ideology_score: handle; question, the short phrase shown with an answer ("Abortion position"); anes_variable, the ANES variable; domain, "demographic" or the attitude block; wording, the question as printed in the ANES codebook; wording_variable and wording_page, the variable whose question that is and the codebook page where it begins; and wording_status. A status of "verbatim" means the wording is the variable's own question; "from_source_item" means the variable is a summary without a question of its own and the wording is that of the item it summarizes (for party identification, V241221); "stem_only" means the codebook prints only a shared stem ("Where would you place yourself on this scale, or haven't you thought much about this?") and the subject of the scale is in its end labels; "none" means no single question exists (census region, age), and wording is NA.

- `demographic_fields`: The handles of the demographic columns.
- `n_demographics`, `n_attitudes`: 17 and 105.
- `source`, `citation`: The citation of the ANES release.
- `build`: The R and package versions, the md5 sums of the two ANES files, the seed, and the label overrides used to build the data.
- `build_audit`: For each column, the number of `NA` cells and the number of missing-data codes in the raw file for the same respondents; the two are equal.

Details

The respondents were drawn from the 4,710 of 5,521 who completed both the pre-election and the post-election interview, neither of them on paper, and answered at least 75 percent of the 122 fields. For each code of party identification, race/ethnicity, and liberal-conservative self-placement not yet covered, one respondent with that code was drawn at random (seed 110); the remaining places were filled by greedy maximin selection, each time adding the respondent farthest from those already chosen, under a Gower distance in which the demographic, identity, thermometer, evaluation, issue, and value blocks carry weights 0.25, 0.25, 0.20, 0.15, 0.15, and 0.15. Each respondent's answers are kept together.

The sample spans unusual combinations of answers and does not reproduce population proportions. It carries no survey weights and supports no population inference. Maximin selection overrepresents people with no 2024 presidential vote recorded (38 of the 100, against 18.5 percent of the eligible respondents), and within each party it favors members whose other answers depart from the party's usual positions.

`ideology_score` comes from a one-dimensional graded-response item response model fitted to 89 attitude items over the 4,710 eligible respondents. Categories enter in the order of their ANES codes (thermometers grouped into five bands, off-scale answers such as "Haven't thought much about this" left out), and items are reversed where they correlate negatively with party identification and ideology. The expected a posteriori scores are standardized over the eligible respondents and oriented so that low is liberal and high is conservative. They correlate 0.89 with party identification and ideology among the eligible respondents and 0.58 among the 100, whose scores have a standard deviation of 0.55.

Items on sexual orientation and gender identity are excluded by design, as are restricted-use variables, geography below the census region, open-ended answers, and items asked only of the spouse/partner sample, only on paper, or only of the GSS re-interview sample, which this release does not include.

Source

American National Election Studies. 2026. ANES 2024 Time Series Study Full Release [dataset and documentation]. May 19, 2026 version. www.electionstudies.org. Work that uses these profiles should cite ANES. ANES bears no responsibility for the analyses or interpretations presented here.

Examples

```
dim(anes_2024_personas)
dictionary <- attr(anes_2024_personas, "dictionary")
dictionary[dictionary$handle == "att_iss_abortion_position",
           c("question", "wording", "wording_status")]
```

```
table(anes_2024_personas$att_id_party_identification)
issues <- anes_2024_personas[, startsWith(names(anes_2024_personas), "att_iss_")]
ncol(issues)
```

`as_persona_frame` *Attach the persona contract to a data frame*

Description

Attaches persona metadata to a decoded data frame. When a `persona_frame` is passed to `panel_from_data()` without a template, demographic fields and stated answers are rendered separately. A question map supplies the wording used for answer fields. Plain data frames use the flat key-value rendering.

Usage

```
as_persona_frame(data, questions = NULL, demographics = NULL, answers = NULL)
```

Arguments

`data`	A data frame, one respondent per row. Columns may hold text, numbers, factors, or labeled survey values as `haven::read_dta()` and `haven::read_sav()` return them; values are decoded when the personas are rendered, by the rules under "Rendering of values" in `panel_from_personas()`.
`questions`	Optional named character vector mapping column names to the human question wording, e.g. `c(pid = "Party identification", ab = "Abortion position")`. Columns absent from this map keep their column name. Without it the column names stand in for the questions, which is formatting, not a faithful translation, and rendering the stated answers draws a warning.
`demographics`	Optional character vector of columns to treat as demographic background (the rest become stated answers). Defaults to the common demographic names found in `data`.
`answers`	Optional character vector restricting which columns may appear as stated answers. Defaults to every column that is not a demographic, an `id`-named, or a `weight`-named column, so analysis-only columns do not leak into the prompt. Identifier columns are never rendered, even when named here.

Value

`data` with the persona contract attached and class `persona_frame`.

See Also

`panel_from_data()`, `panel_from_personas()`, `LLMR::llm_persona_split()`.

Examples

```
df <- data.frame(
  age = c("35-44", "65+"),
  pid = c("Strong Democrat", "Strong Republican"),
  ab  = c("Always legal", "Never legal"))
pf <- as_persona_frame(
  df,
  questions = c(pid = "Party identification", ab = "Abortion position"),
  demographics = "age")
```

`conjoint_amce` *AMCEs from a conjoint administration*

Description

Average marginal component effects from a `conjoint_instrument()` administration: one OLS regression of profile choice on treatment-coded dummies for all attributes simultaneously, with CR1 cluster-robust standard errors clustered by persona and 95% intervals on the t distribution with G - 1 degrees of freedom (G personas). Under uniform, independent profile randomization this is the standard AMCE estimator. The regression uses the respondent-level profiles recorded during administration, not the profiles in the initial design table.

Usage

```
conjoint_amce(responses)
```

Arguments

`responses`	A `panel_administer()` result whose instrument came from `conjoint_instrument()`.

Details

Each attribute's effects are relative to its reference level: the first level, in the design's order, that appears in the estimation data. That is the first declared level unless every profile showing it was lost, for example because the executions that drew it failed or their replies did not parse. A warning names the attribute and the level that took its place when this happens, and also names any other declared level that is absent and so has no row in the table.

Value

A `conjoint_amce` tibble: `attribute`, `level`, `estimate`, `std_error`, `ci_lo`, `ci_hi`. Reference levels appear with estimate 0 and `std_error` = `NA`, so the table feeds the familiar conjoint plot directly. The ordinary columns `n_profiles`, `n_respondents`, `n_dropped_na`, and `n_execution_failures` record the profile rows used, the respondents administered (including those whose responses failed), missing task responses dropped, and failed executions. The attribute `reference` is a character vector naming each attribute's reference level, and `n_clusters` is the number of personas with at least one estimated response, the clusters behind the standard errors.

References

Hainmueller, Jens, Daniel J. Hopkins, and Teppei Yamamoto (2014). "Causal Inference in Conjoint Analysis: Understanding Multidimensional Choices via Stated Preference Experiments." *Political Analysis* 22(1), 1-30.

Examples

```
set.seed(110)
panel <- panel_from_margins(list(group = c(A = .5, B = .5)), n = 6)
design <- conjoint_design(
  list(color = c("blue", "red"), cost = c("low", "high")),
  n_tasks = 6)
instrument <- conjoint_instrument(design)
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b")

# A synthetic respondent who always takes the red profile: the AMCE for red
# is recovered, and cost is flat.
prefers_red <- function(experiments, ...) {
  shown <- vapply(experiments$messages, `[[`, "", "user")
  first <- sub("Profile 2.*", "", shown)
  experiments$response_text <- ifelse(grepl("red", first),
                                      "Profile 1", "Profile 2")
  experiments
}
r <- panel_administer(panel, instrument, cfg, .runner = prefers_red)
conjoint_amce(r)


if (nzchar(Sys.getenv("GROQ_API_KEY"))) {
  conjoint_amce(panel_administer(panel, instrument, cfg))
}
```

`conjoint_design` *Conjoint tasks*

Description

Random profile pairs (or k-tuples) over the supplied attributes, the design for a forced-choice conjoint. Profiles are sampled uniformly and independently per attribute; set a seed beforehand for a reproducible design (the function never sets one). At administration, fresh profiles are drawn independently for every respondent from the same attribute levels.

Usage

```
conjoint_design(attributes, n_tasks = 5L, profiles_per_task = 2L)
```

Arguments

attributes	Named list of level vectors, with at least two attributes. Attribute names must be unique and nonempty and must not be task, profile, persona_id, or chosen, which name the design's own columns. Each attribute needs at least one level, and its levels must be unique, nonempty, and not missing.
n_tasks	Tasks per respondent: one whole number of at least 1.
profiles_per_task	Profiles shown per task (default 2): one whole number of at least 1. conjoint_instrument() needs at least two.

Details

Within a task, a profile that repeats one already drawn is redrawn, up to 25 times. When the attribute space holds fewer distinct profiles than profiles_per_task, every task repeats a profile. Otherwise a repeat remains only when all 25 redraws repeat a profile, which can happen when the space is barely larger than the task. Either case draws a warning that says which occurred. The profiles drawn for each respondent at administration follow the same rule without a warning.

Value

A conjoint_design list with fields profiles, a tibble containing task, profile, and one column per attribute, and attributes, the named list of attribute levels. Render it into forced-choice items with conjoint_instrument() and estimate with conjoint_amce() after administration.

Examples

```
set.seed(110)
conjoint_design(
  list(price = c("$10", "$20"), speed = c("slow", "fast")),
  n_tasks = 4)
```

---

conjoint_instrument *Build a conjoint instrument*

Description

Converts a conjoint design into one forced-choice item per task. Administration draws profiles for each persona and asks the model to choose a profile by its label.

Usage

```
conjoint_instrument(design, question = "Which profile do you prefer?")
```

Arguments

design	A conjoint_design() object.
question	Question text shown above each task's profiles.

Details

Each persona-task pair is a separate request. Its `item_position` is instrument metadata and does not describe an order of exposure. Profile choice options are permuted for each request.

Profile descriptions follow profile-number order. Attributes within a description follow the design's attribute order. The initial profile table in `design` is not reused across personas.

Value

A `panel_instrument` with task identifiers such as `task_1` and profile options such as `"Profile 1"`. Each item retains the attribute levels used for profile draws. The instrument's `conjoint` field retains the design metadata used by `conjoint_amce()`.

Examples

```
set.seed(110)
panel <- panel_from_margins(list(group = c(A = 1)), n = 4)
design <- conjoint_design(
  list(economy = c("weak", "strong"), taxes = c("lower", "higher")),
  n_tasks = 3
)
instrument <- conjoint_instrument(
  design, "Which candidate do you prefer?"
)
cfg <- LLMR::llm_config("groq", "example-model")
runner <- function(experiments, ...) {
  experiments$response_text <- "Profile 1"
  experiments
}
panel_administer(panel, instrument, cfg, .runner = runner)
```

`panel_administer` *Administer an instrument to a panel*

Description

Creates one independent request for each persona-item pair. The persona is placed in the system message, and the item and its options are placed in the user message.

Usage

```
panel_administer(
  panel,
  instrument,
  config,
  max_calls = 5000L,
  confirm = FALSE,
  price_table = NULL,
  tokens_per_call = NULL,
```

```
  .runner = NULL,
  ...
)
```

Arguments

panel	A result from panel_from_margins(), panel_from_data(), or panel_from_personas().
instrument	A panel_instrument() object.
config	An LLMR::llm_config() for a generative model.
max_calls	Maximum planned request count before confirmation is required. The default is 5000.
confirm	Set TRUE to proceed when the request count exceeds max_calls.
price_table, tokens_per_call	Optional inputs for a preflight cost calculation. Supply both. price_table uses the LLMR::llm_usage() format, with model, input, and output columns and prices per million tokens. A single tokens_per_call value gives a total priced over the possible input-output allocation. Two values, c(input, output), specify the allocation. The package supplies neither prices nor estimated token counts.
.runner	Optional function (experiments, ...) for offline or external execution. It receives a data frame with config and messages list-columns and returns the experiment rows with request_id and response_text. Each submitted request identifier must occur once. Returned rows may be reordered. The default uses LLMR::call_llm_par().
...	Passed to the runner.

Details

Closed-item replies are matched to the offered options. Unmatched replies receive NA. Open-item replies retain their text with surrounding whitespace removed.

Requests do not contain earlier items or responses. Instrument position therefore does not describe an order of exposure.

With "option_order" randomization, choice options are permuted for each request. Likert scales retain their supplied order or are reversed, each with probability one-half. Likert scores always use the supplied scale order.

Value

A panel_responses object.

- data contains one row per persona-item pair. Its identifiers are persona_id, item_id, and type.
- item_position is the item's fixed, 1-based instrument position. option_order records displayed options separated by |.
- response contains the matched option or open-item text. score gives the 1-based position on a Likert item's supplied scale.

- Execution columns retain `response_text`, `response_id`, `success`, `error_message`, `finish_reason`, `model`, and `provider`. Request hashes, model versions, and creation times are retained when supplied by the runner.
- Conjoint administrations include recorded profile draws in a `profiles` list-column.
- `panel` and `instrument` retain the supplied study objects. `benchmark` is `NULL` until `panel_benchmark()` is called.
- `usage` retains available token or duration records and their execution identifiers. It is `NULL` when no usage fields were returned.

Examples

```
set.seed(110)
panel <- panel_from_margins(
  list(party = c(left = .5, right = .5)), n = 6
)
instrument <- panel_instrument(
  item_likert("wk4", "A four-day work week would benefit society.")
)
cfg <- LLMR::llm_config("groq", "example-model")
runner <- function(experiments, ...) {
  experiments$response_text <- "agree"
  experiments
}
responses <- panel_administer(
  panel, instrument, cfg, .runner = runner
)
responses$data
```

`panel_batch_fetch` *Fetch and parse a completed panel batch job*

Description

Retrieves the batch results and parses them into a `panel_responses`, identical in shape to a synchronous `panel_administer()` run. Responses are joined to the grid by request id, so the order the provider returns them in does not matter.

Usage

```
panel_batch_fetch(job)
```

Arguments

`job`	A `panel_batch_submit()` handle (or a `state_path` to one).

Value

A `panel_responses` object.

See Also

panel_batch_submit(), panel_batch_status().

Examples

```
# Fetching reads the job saved at submission, once the provider reports it
# complete; the result has the same shape as a synchronous run.
state <- file.path(tempdir(), "panel_job.rds")

if (file.exists(state)) {
  responses <- panel_batch_fetch(state)
}
```

---

panel_batch_status *Check the status of a panel batch job*

Description

Check the status of a panel batch job

Usage

```
panel_batch_status(job)
```

Arguments

job	A panel_batch_submit() handle (or a state_path to one).

Value

The LLMR batch status (a one-row tibble).

See Also

panel_batch_submit(), panel_batch_fetch().

Examples

```
# A job saved by panel_batch_submit(state_path = ) is read back by path,
# so its progress can be checked from a later session.
state <- file.path(tempdir(), "panel_job.rds")

if (file.exists(state)) {
  panel_batch_status(state)
}
```

panel_batch_submit *Administer a panel asynchronously through the batch API*

Description

Submits one request per persona and item to a provider's batch API and returns a job handle. Use panel_batch_status() to inspect the job and panel_batch_fetch() to retrieve completed results. Provider services determine prices and completion times.

Usage

```
panel_batch_submit(
  panel,
  instrument,
  config,
  state_path = NULL,
  max_calls = 5000L,
  confirm = FALSE
)
```

Arguments

panel	A panel_from_margins() / panel_from_data() / panel_from_personas() result.
instrument	A panel_instrument().
config	An LLMR::llm_config() for a generative model on a provider with a supported batch API (OpenAI, Groq, Anthropic, Gemini).
state_path	Optional path; when given the job is also saved there as RDS so it can be fetched from another session.
max_calls	Integer. If the run would make more than this many calls, it stops unless confirm = TRUE. Default 5000.
confirm	Logical. Set TRUE to proceed past max_calls.

Details

All personas are administered under one config (one model). The handle carries the survey prompts and rendered persona text. When state_path is supplied, the API key must be referenced through an environment variable so its value is not written to the saved state.

Value

A panel_batch_job handle.

See Also

panel_batch_fetch(), panel_batch_status(), panel_administer().

Examples

```
# Three personas and one item make a batch of three requests.
panel <- panel_from_margins(list(party = c(left = .5, right = .5)), n = 3)
instrument <- panel_instrument(item_likert("wk4", "A four-day work week helps."))
cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b",
                        api_key = LLMR::llm_api_key_env("GROQ_API_KEY"))
state <- file.path(tempdir(), "panel_job.rds")


if (nzchar(Sys.getenv("GROQ_API_KEY"))) {
  job <- panel_batch_submit(panel, instrument, cfg, state_path = state)
  panel_batch_status(job)
}
```

`panel_benchmark` *Compare panel responses with a human benchmark*

Description

Compares closed-item response shares with human benchmark shares. The comparison uses responses that parsed and were not reported as execution failures. It does not alter responses or adjust their shares.

Usage

```
panel_benchmark(responses, benchmark, benchmark_name = "benchmark")
```

Arguments

`responses`	A `panel_administer()` result.
`benchmark`	A data frame with `item_id`, `response`, and `share` columns. Shares must be numeric probabilities in `[0, 1]` without missing values. The function raises an error when an item's share total differs from 1 by more than 0.01. Each item-response pair must occur once, and covered response labels must be offered options. An offered option that the benchmark omits for a covered item is taken as a known human share of zero, not as a missing value.
`benchmark_name`	Name used to identify the benchmark in reports.

Details

Without a benchmark, response shares describe the configured model under the supplied personas.

The function refuses conjoint responses because profile assignments differ across personas. It also refuses a covered item when every execution failed. If executions occurred but no response parsed for an item, its panel shares and deviations are `NA`.

Value

responses with its benchmark field populated.

- table contains the panel share, human share, and their difference for each covered item-response pair.
- nonresponse contains execution-failure and nonresponse rates by item.
- items_covered and items_total record closed-item coverage.
- mean_abs_dev and max_dev summarize absolute deviations.
- benchmark_name and ts identify the comparison and its time.

Examples

```
panel <- panel_from_margins(list(group = c(A = 1)), n = 4)
instrument <- panel_instrument(
  item_choice("fund", "Choose an investment.", c("transit", "roads")),
  randomize = character(0)
)
cfg <- LLMR::llm_config("groq", "example-model")
runner <- function(experiments, ...) {
  experiments$response_text <- "transit"
  experiments
}
responses <- panel_administer(
  panel, instrument, cfg, .runner = runner
)
benchmark <- data.frame(
  item_id = "fund",
  response = c("transit", "roads"),
  share = c(.6, .4)
)
panel_benchmark(responses, benchmark, "Example human benchmark")
```

---

panel_bias_audit *Summarize execution failures, parse failures, and first-option sensitivity*

Description

Counts execution and parse failures by item. For closed items administered with randomized option order, it also applies a chi-squared test to the chosen response and the option shown first. The test does not use the full option permutation.

Usage

```
panel_bias_audit(responses)
```

Arguments

responses	A panel_administer() result.

Value

A tibble: item_id, n, parse_failures, execution_failures, order_effect_p, and order_test_note. order_effect_p is the first-option chi-squared p-value. When it is NA, order_test_note says why: the item is open, its option order did not vary, fewer than four responses parsed, every parsed response is the same, one option was shown first in every parsed response, the expected cell counts are below five, or the test itself failed, in which case a warning is also issued. order_test_note is NA when a p-value is reported.

Examples

```
panel <- panel_from_margins(list(group = c(A = 1)), n = 4)
instrument <- panel_instrument(
  item_choice("pick", "Choose one.", c("A", "B")),
  randomize = character(0))
config <- LLMR::llm_config("groq", "example-model")
runner <- function(experiments, ...) {
  experiments$response_text <- "A"
  experiments$success <- TRUE
  experiments
}
responses <- panel_administer(panel, instrument, config, .runner = runner)
panel_bias_audit(responses)
```

---

panel_from_data *Draw a persona panel from microdata rows*

Description

Samples rows from a data frame with replacement, which preserves the joint distribution of the selected attributes, and renders each sampled row as a persona. This is the joint-distribution counterpart of panel_from_margins(), which samples attributes independently. The margins the report cites are computed from the source data, one prop.table(table()) per selected column over its values as the personas state them, so a missing-data label is not a category.

Usage

```
panel_from_data(
  data,
  n,
  persona_template = NULL,
  columns = NULL,
  weights = NULL,
  na_strings = NULL
)
```

Arguments

`data`	A data frame, one row per source case.
`n`	Panel size.
`persona_template`	Text with `{attribute}` placeholders rendered per persona. `NULL` builds a plain "attribute: value" persona. A placeholder whose value is left out as missing is filled with nothing, and a placeholder may not name an identifier or the `weights` column.
`columns`	Attribute columns to keep. Defaults to every column except the `weights` column when one is given. Identifier columns are kept in the panel but never rendered into the persona text.
`weights`	Optional name of a single column of nonnegative sampling weights (rows are drawn with probability proportional to it). The column is never rendered into the persona text.
`na_strings`	Regular expressions, matched ignoring case against each value and value label, that mark it as missing. `NULL` (the default) uses the patterns of FocusGroup's `fg_default_na_strings()`, which match the missing-data labels of the ANES and similar releases: "Inapplicable", "Refused", "Don't know", "Not asked", "DK/RF", "No post-election interview", "Insufficient partial, interview deleted", "RESTRICTED", "Error", "Technical error", "Not reported", and the like. `character(0)` treats no label as missing. Answers that use such words ("2. Refused to take sides", "I keep missing the bus") are kept.

Details

Without a template, a plain data frame renders as "attribute: value" pairs, and a `persona_frame` (see `as_persona_frame()`) renders as background and stated answers, as `panel_from_personas()` does. Values are written as text by the rules under "Rendering of values" in `panel_from_personas()`.

For a reproducible panel, set a seed before calling (the function never sets one itself).

Value

A `silicon_panel`: a tibble with `persona_id`, the selected attribute columns, and `persona`. Its `source_rows` attribute gives, for each persona in `persona_id` order, the row of `data` it was drawn from.

Examples

```
set.seed(110)
src <- data.frame(
  education = c("college", "college", "no college", "no college"),
  income    = c("high", "high", "low", "low"),
  weight    = c(2, 2, 1, 1))
panel_from_data(src, n = 10, columns = c("education", "income"),
                weights = "weight",
                persona_template = "A {education} respondent earning {income}.")
```

`panel_from_margins` *Draw a persona panel from population margins*

Description

Samples `n` personas with attributes drawn independently from the supplied margins, and renders each persona's text from a template. Attributes are sampled independently, so the result does not preserve their joint distribution. Use `panel_from_data()` to sample complete microdata rows.

Usage

```
panel_from_margins(margins, n, persona_template = NULL)
```

Arguments

`margins`	A named list; each element a named probability vector, e.g. `list(age = c("18-34" = .3, "35-64" = .45, "65+" = .25))`. Probabilities are renormalized if they do not sum to 1.
`n`	Panel size.
`persona_template`	Text with `{attribute}` placeholders rendered per persona. `NULL` builds a plain "attribute: value" persona.

Details

For a reproducible panel, set a seed before calling (the function never sets one itself).

Value

A `silicon_panel`: a tibble with `persona_id`, one column per attribute, and `persona` (the rendered text).

Examples

```
set.seed(110)
panel <- panel_from_margins(
  list(cohort = c(young = .3, middle = .45, older = .25),
       party  = c(left = .45, right = .45, independent = .10)),
  n = 50,
  persona_template = "A {cohort} voter who leans {party}."
)
panel
```

panel_from_personas *Draw a panel from a persona data frame*

Description

Turns rows of a persona data frame (one respondent per row, demographics plus survey or attitude answers) into a `silicon_panel` whose personas can be administered survey items. It is built for frames following the LLMR persona contract, such as the bundled anes_2024_personas: the demographic fields are those that `LLMR::llm_persona_demographic_fields()` names, every other column is a stated answer, answers are shown under their question wording when the frame carries a dictionary or `questions` gives one, and each persona is rendered as a person to answer in character ("You are this person: ... On a questionnaire you gave these answers: ...").

Usage

```
panel_from_personas(
  data,
  n = NULL,
  rows = NULL,
  weights = NULL,
  questions = NULL,
  na_strings = NULL
)
```

Arguments

`data`	A persona data frame, such as anes_2024_personas. Those 100 rows are diversity-selected United States respondents and carry no survey weights, so they stand for no population.
`n`	Optional panel size. With `NULL`, every selected row is used; with a number, rows are sampled (without replacement when `n` does not exceed the pool, otherwise with replacement).
`rows`	Optional row selector: a numeric vector of whole-number row positions, a logical vector with one entry per row, or a predicate `function(df)` returning such a logical vector. Applied before sampling. Character and factor selectors are refused, since a factor's codes are not row positions.
`weights`	Optional survey weights for the draw: a column name in `data`, or a numeric vector aligned to the selected rows. Used only when `n` is given. `NULL` (default) draws uniformly. A weight column is never rendered into the persona text.
`questions`	Optional named character vector of question texts keyed by column name, as in `c(services = "Should the government provide fewer or more services?")`. It takes precedence over the question in the frame's dictionary. Names that are not columns of `data` draw a warning.
`na_strings`	Regular expressions, matched ignoring case against each value and value label, that mark it as missing. `NULL` (the default) uses the patterns of FocusGroup's `fg_default_na_strings()`, which match the missing-data labels of

the ANES and similar releases: "Inapplicable", "Refused", "Don't know", "Not asked", "DK/RF", "No post-election interview", "Insufficient partial, interview deleted", "RESTRICTED", "Error", "Technical error", "Not reported", and the like. `character(0)` treats no label as missing. Answers that use such words ("2. Refused to take sides", "I keep missing the bus") are kept.

Details

Unlike `panel_from_margins()` and `panel_from_data()`, this constructor keeps each selected respondent's answers together. The `margins` attribute contains the demographic distribution of the selected rows, and the `source_rows` attribute gives, for each persona in `persona_id` order, the row of `data` it was drawn from. A persona can therefore be traced to its source record with `attr(panel, "source_rows")[panel$persona_id]`, which still holds after the panel's rows are subset.

For a reproducible draw, set a seed before calling (the function never sets one itself).

Value

A `silicon_panel`: a tibble with `persona_id`, the demographic columns as the persona text gives them (`NA` where a value is left out), and `persona`, with the attributes `margins`, `source`, and `source_rows`.

Rendering of values

Each value of a rendered field is written as text by these rules.

- A labeled column, such as `haven::read_dta()` and `haven::read_sav()` return, is decoded from its `labels` attribute; haven itself is not needed. A labeled code is shown as its label without the code prefix ("1. Male" becomes "Male"). A code without a label on a scale labeled only at its ends is shown with the scale, as in "5 on a scale from 1 (Fewer services) to 7 (More services)"; on a scale centered on zero the ends are the labeled codes -m and m, even when its midpoint is labeled too. A negative code without a label is left out as missing, with a warning. Any other code without a label is shown as the code, with a warning, unless the column labels at most one nonnegative code or only top and bottom codes ("80. Age 80 or older"), in which case the column holds numbers and the value is shown as one. Values that an SPSS file declares missing (`na_values`, `na_range`) are left out.
- A factor's levels are its labels. Levels as `haven::as_factor()` writes them ("1. Fewer services", "[1] 1. Fewer services", and bare numbers for unlabeled codes) are decoded by the rules above; a level without a code prefix ("Refused" from a file whose labels carry none) counts as a label whose code is unknown. A character column that mixes coded labels with bare numbers, as `as.character(haven::as_factor())` writes it, is decoded the same way. Code prefixes are removed from factor levels and from character values.
- `NA`, empty values, and missing-data labels are left out. A value or a value label is one when it matches `na_strings` and either carries a missing-data code (negative, or 90 and above: "-9 Refused", "998. Don't know") or says nothing but the matched wording once codes, interviewer markers, and asides in parentheses are set aside ("Refused to answer", "Don't know (VOL)"), as FocusGroup decides, so a free-text answer such as "I refused to answer the door" is kept. A value with a negative code prefix that does not match `na_strings` is shown, with a warning, since many releases use negative codes for missing data; the labels of negative codes on a scale centered on zero ("-3. Far left" beside "3. Far right") draw none.

- Numbers are written in full: 100000, not 1e+05.
- A label exactly 80 characters long draws a warning. Stata files cut value labels at 80 characters, and the ANES 2024 release has labels cut there.
- Identifier columns (named, ignoring case, `id`, `caseid`, `case_id`, `respondent_id`, `respid`, `resp_id`, `persona_id`, `uid`, `record_id`, or `row_id`, or with a variable label that calls them a case or respondent ID, as the ANES label of V240001 does), the column named by `weights`, a column named `persona`, and `ideology_score` are never rendered.

The values left out as missing are reported in one message per call, naming each field and the number of personas whose text omits it. A stated answer shown under its column name, for want of a question wording, draws a warning.

See Also

anes_2024_personas, `as_persona_frame()`, `panel_administer()`.

Examples

```
set.seed(110)
panel <- panel_from_personas(anes_2024_personas, n = 8)
panel

# A labeled column, as haven::read_dta() returns it, is decoded from its
# labels: the unlabeled 5 is placed on its scale and -9 is left out.
survey <- data.frame(age = c("18-29", "65 or older", "30-49"))
survey$services <- structure(c(1, 5, -9), labels = c(
  "1. Government should provide many fewer services" = 1,
  "7. Government should provide many more services" = 7,
  "-9. Refused" = -9))
labeled_panel <- panel_from_personas(
  survey, questions = c(services = "Government services and spending"))
labeled_panel$persona
```

`panel_generics` *Shared generic methods*

Description

LLMRpanel provides `LLMR::diagnostics()`, `LLMR::report()`, and `plot()` methods for `panel_responses` objects. It provides `tibble::as_tibble()` methods for `panel_responses` and `silicon_panel` objects.

Value

`diagnostics()` returns the `panel_bias_audit()` tibble, one row per item, with four columns that repeat on every row: `benchmark_state` (`"NOT BENCHMARKED"`, `"PARTIALLY BENCHMARKED"`, or `"BENCHMARKED"`), `items_covered` and `items_total`, the numbers of closed items covered by the benchmark and administered, and `mean_abs_dev`, the benchmark's mean absolute deviation, or `NA` when there is no benchmark.

`report()` returns a `panel_report` object: a character vector that prints one line per element. The lines give the benchmark state, the panel's source and demographic fields, the response and failure counts, and each item's first-option sensitivity.

`as_tibble()` returns a tibble. For `panel_responses` it is the `data` field, one row per persona-item pair. For a `silicon_panel` it is the persona table without the `silicon_panel` class and without the `margins`, `source`, and `source_rows` attributes.

`panel_instrument` *Assemble an instrument*

Description

Combines survey items and records the option-order randomization used during administration.

Usage

```
panel_instrument(items, randomize = "option_order")
```

Arguments

`items`	A list of panel_items, or a single item. Item identifiers must be unique.
`randomize`	`"option_order"` by default. Choice options are permuted for each response. A Likert scale retains its supplied order or is reversed, each with probability one-half. Use `character(0)` to retain the supplied option order.

Details

Each persona-item pair is administered as an independent request. The model does not receive earlier items or responses. `"item_order"` is therefore refused.

Administration records the displayed options in `option_order`. `item_position` is the item's fixed, 1-based position in the instrument. Likert scores use the supplied scale order, including when the displayed scale is reversed.

Value

A `panel_instrument` object containing `items` and `randomize`.

Examples

```
panel_instrument(list(
  item_likert("trust", "The city council acts in the public interest."),
  item_open("reason", "What would improve the city council's work?")
))

panel_instrument(
  item_choice("fund", "Choose an investment.", c("transit", "roads")),
  randomize = character(0)
)
```

panel_items *Survey items*

Description

Three item types cover most quantitative instruments: a Likert item (an agree-disagree battery row), a forced choice, and an open item (free text, returned verbatim). Likert responses also get a numeric `score` (position on the scale as given, 1-based).

Usage

```
item_likert(
  id,
  text,
 scale = c("strongly disagree", "disagree", "neutral", "agree", "strongly agree")
)

item_choice(id, text, options)

item_open(id, text)
```

Arguments

`id`	Item identifier: one nonempty string, unique within an instrument.
`text`	The question text: one nonempty string.
`scale`	For `item_likert()`: two or more response labels from low to high.
`options`	For `item_choice()`: two or more choice labels.

Details

Replies are matched to the labels after trimming surrounding spaces and, failing an exact match, without regard to case. Labels must therefore differ by more than case or surrounding spaces; labels that do not, and missing or empty labels, are refused when the item is built.

Value

An object of class `panel_item`.

Examples

```
item_likert("wk4", "A four-day work week would benefit society.")
item_choice("vote", "Which proposal do you prefer?", c("A", "B"))
item_open("why", "What would improve the city council's work?")
```

panel_usage *Usage diagnostics for an administered panel*

Description

Summarizes token and outcome diagnostics recorded by `panel_administer()` or `panel_batch_fetch()`. The diagnostics are stored in the `usage` field of a `panel_responses` object and summarized by `LLMR::llm_usage()`. Model and provider remain in the returned frame. A supplied `price_table` adds a cost column. The package contains no price table. When the runner records per-call `duration`, its sum is returned as `duration_s`.

Usage

```
panel_usage(responses, price_table = NULL)
```

Arguments

`responses`	A `panel_administer()` result.
`price_table`	Optional price table passed to `LLMR::llm_usage()`.

Value

A one-row usage tibble, or a typed empty tibble when the runner returned no recorded usage fields.

See Also

`panel_administer()`, `LLMR::llm_usage()`.

Examples

```
panel <- panel_from_margins(list(group = c(A = 1)), n = 2)
instrument <- panel_instrument(
  item_choice("pick", "Choose one.", c("A", "B")),
  randomize = character(0))
config <- LLMR::llm_config("groq", "example-model")
runner <- function(experiments, ...) {
  experiments$response_text <- "A"
  experiments$sent_tokens <- 4L
  experiments$rec_tokens <- 1L
  experiments$total_tokens <- 5L
  experiments$success <- TRUE
  experiments
}
responses <- panel_administer(panel, instrument, config, .runner = runner)
panel_usage(responses)
```

`plot.panel_responses` *Plot a benchmark comparison*

Description

Plots the comparison recorded by `panel_benchmark()`. Each covered response level has one point for the panel share and one for the benchmark share, joined by a segment. Response levels follow the instrument's option order, and items appear in separate panels. The method requires a benchmark record.

Usage

```
## S3 method for class 'panel_responses'
plot(x, ...)
```

Arguments

`x`	A `panel_administer()` result that `panel_benchmark()` has been run on (the comparison is stored in `x$benchmark`).
`...`	Ignored; reserved for generic dispatch.

Value

A ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(110)
  panel <- panel_from_margins(list(party = c(left = .5, right = .5)),
                              n = 12,
                              persona_template = "A voter who leans {party}.")
  instrument <- panel_instrument(
    item_choice("plan", "Which plan do you prefer?", c("Plan A", "Plan B")))
  cfg <- LLMR::llm_config("groq", "openai/gpt-oss-20b")
  by_party <- function(experiments, ...) {
    experiments$response_text <- ifelse(
      grepl("leans left", vapply(experiments$messages, `[[`, "", "system")),
      "Plan A", "Plan B")
    experiments
  }
  r <- panel_administer(panel, instrument, cfg, .runner = by_party)
  bench <- data.frame(item_id = "plan",
                      response = c("Plan A", "Plan B"),
                      share = c(.55, .45))
  plot(panel_benchmark(r, bench, "city survey 2025"))
}
```

`run_panel_studio` *Launch the LLMRpanel Shiny GUI*

Description

Starts a Shiny application that builds a persona panel, administers a choice item or conjoint instrument, and presents the package's diagnostics and design analyses. Choice-item response shares can be compared with an optional benchmark. The application can download the responses and report in a zip file, with the benchmark table when one is available.

Usage

```
run_panel_studio(...)
```

Arguments

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

Details

The GUI is optional. It needs the suggested packages `shiny`, `bslib`, `DT`, and `LLMR.shiny`; install them first. Keys are read from environment variables only, never pasted into the app; a deterministic demo mode runs offline.

Value

Invisibly, the value of `shiny::runApp()`; called for the side effect of starting the app.

Examples

```
if (interactive() &&
    requireNamespace("shiny", quietly = TRUE) &&
    requireNamespace("LLMR.shiny", quietly = TRUE)) {
  run_panel_studio()
}
```

Index

∗ **datasets**
anes_2024_personas, 2

anes_2024_personas, 2, *19*, *21*
as_persona_frame, 5
as_persona_frame(), *17*, *21*
as_tibble.panel_responses *(*panel_generics*)*, 21
as_tibble.silicon_panel *(*panel_generics*)*, 21

conjoint_amce, 6
conjoint_amce(), *8, 9*
conjoint_design, 7
conjoint_design(), *8*
conjoint_instrument, 8
conjoint_instrument(), *6*, *8*

diagnostics.panel_responses *(*panel_generics*)*, 21

item_choice *(*panel_items*)*, 23
item_choice(), *23*
item_likert *(*panel_items*)*, 23
item_likert(), *23*
item_open *(*panel_items*)*, 23

LLMR::call_llm_par(), *10*
LLMR::llm_persona_demographic_fields(), *19*
LLMR::llm_persona_split(), *5*
LLMR::llm_usage(), *10*, *24*

panel_administer, 9
panel_administer(), *6*, *11*, *13, 14*, *16*, *21*, *24, 25*
panel_batch_fetch, 11
panel_batch_fetch(), *12, 13*, *24*
panel_batch_status, 12
panel_batch_status(), *12, 13*
panel_batch_submit, 13
panel_batch_submit(), *11, 12*
panel_benchmark, 14
panel_benchmark(), *11*, *25*
panel_bias_audit, 15
panel_bias_audit(), *21*
panel_from_data, 16
panel_from_data(), *5*, *10*, *13*, *18*, *20*
panel_from_margins, 18
panel_from_margins(), *10*, *13*, *16*, *20*
panel_from_personas, 19
panel_from_personas(), *2*, *5*, *10*, *13*, *17*
panel_generics, 21
panel_instrument, 22
panel_instrument(), *10*, *13*
panel_items, *22*, 23
panel_usage, 24
plot.panel_responses, 25

report.panel_responses *(*panel_generics*)*, 21
run_panel_studio, 26

shiny::runApp(), *26*