Package: surveyframe (via r-universe)

October 30, 2026

Title Survey Instrument Workflows

Version 0.4.3

Description Provides a design-first survey research workflow. An instrument, an analysis plan declared before data collection, and a measurement or structural model are held together in one typed, integrity-checked object (the 'sframe'), so a study's confirmatory tests are fixed before responses arrive rather than chosen afterward. Includes visual instrument design via a browser-based builder or 'Shiny' studio, export to a self-contained static HTML survey, an embeddable 'Shiny' module, SHA-256 integrity-checked serialisation to the '.sframe' format, multi-page survey rendering with branching logic, response quality checking, scale scoring, psychometric diagnostics, analysis-plan execution, model syntax generation for EFA, CFA, CB-SEM, and PLS-SEM, an interactive response dashboard, codebook generation, and reproducible HTML reporting. Also supports multi-criteria decision analysis (AHP, ANP, DEMATEL, TOPSIS, VIKOR, MOORA, SMART, WASPAS, PROMETHEE II, ELECTRE I), small-sample survey helpers, and text and open-ended response analysis (term and n-gram frequency, keyword in context, co-occurrence and co-occurrence networks, sentiment, document-feature matrices, and topic modelling via LDA or structural topic models).

License MIT + file LICENSE

URL https://mohammedalisharafuddin.github.io/surveyframe/, https://github.com/MohammedAliSharafuddin/surveyframe

BugReports https://github.com/MohammedAliSharafuddin/surveyframe/issues

Encoding UTF-8

Language en-GB

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports jsonlite (>= 1.8.0), rlang (>= 1.1.0), openssl (>= 2.1.0)

Suggests ggplot2 (>= 3.4.0), googlesheets4 (>= 1.1.0), shiny (>= 1.7.0), psych (>= 2.3.0), MASS, nnet, logistf (>= 1.24.0), digest (>= 0.6.0), lavaan (>= 0.6-0), seminr (>= 2.3.0), testthat (>= 3.0.0), knitr, rmarkdown, naniar (>= 1.0.0), pagedown (>= 0.20), RMCDA (>= 0.3.1), rstudioapi (>= 0.13), jmv (>= 2.4.0), tidytext (>= 0.4.0), quanteda (>= 3.0.0), stm (>= 1.3.0), topicmodels, igraph (>= 1.5.0), V8 (>= 4.0.0), haven (>= 2.5.0), svglite, callr, chromote, httpuv, pkgload

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-10-30 19:27:53 UTC

RemoteUrl https://github.com/mohammedalisharafuddin/surveyframe

RemoteRef HEAD

RemoteSha b098dcdaf93497a8b9264ac1dd8b92aafd327ba8

Contents

surveyframe-package 6
add_model 8
add_references 8
amend_sframe 9
amendment_log 11
analysis_syntax 12
as.data.frame.sf_choices 14
as.data.frame.sframe 14
as.data.frame.sframe_analysis_results 15
as.data.frame.sframe_assumption_report 16
as.data.frame.sframe_codebook 16
as.data.frame.sframe_descriptives_report 17
as.data.frame.sframe_efa_report 18
as.data.frame.sframe_efa_solution 18
as.data.frame.sframe_item_report 19
as.data.frame.sframe_missing_data_report 20
as.data.frame.sframe_quality_report 20
as.data.frame.sframe_reliability_report 21
as.data.frame.sframe_sample_size_plan 22
as.data.frame.sframe_sensitivity 22
as.data.frame.sframe_validation 23
as.data.frame.sframe_validity_report 24
as_sframe 24
assumption_report 25
bootstrap_ci 26

cfa_lavaan_syntax . . . 27
cfa_syntax . . . 28
clean_text_responses . . . 29
codebook_report . . . 30
cohens_d_ci . . . 31
cramers_v_ci . . . 32
descriptives_report . . . 33
efa_report . . . 34
efa_solution . . . 35
efa_syntax . . . 36
eta_sq_ci . . . 37
export_google_sheet . . . 38
export_static_survey . . . 39
extract_quotes . . . 42
format.sf_branch . . . 43
format.sf_check . . . 43
format.sf_choices . . . 44
format.sf_item . . . 44
format.sf_model . . . 45
format.sf_scale . . . 45
format.sframe . . . 46
item_report . . . 46
launch_builder . . . 47
launch_builder_demo . . . 48
launch_dashboard . . . 49
launch_dashboard_demo . . . 51
launch_studio . . . 51
launch_studio_demo . . . 53
link_git_commit . . . 54
missing_data_report . . . 55
model_json . . . 55
model_report_template . . . 56
ngram_frequency . . . 57
outlier_report . . . 58
plot.sframe_analysis_results . . . 59
posthoc_report . . . 59
preview_demo . . . 60
print.sf_branch . . . 61
print.sf_check . . . 62
print.sf_choices . . . 62
print.sf_item . . . 63
print.sf_model . . . 64
print.sf_scale . . . 64
print.sframe . . . 65
quality_report . . . 65
read_bibtex . . . 67
read_responses . . . 68
read_sframe . . . 70

read_sheet_responses . . . 72
reliability_report . . . 73
render_report . . . 74
render_result . . . 77
render_results . . . 78
render_survey . . . 80
run_analysis_plan . . . 81
sample_size_plan . . . 83
score_scales . . . 84
sem_lavaan_syntax . . . 85
seminr_syntax . . . 86
sensitivity_analysis . . . 87
sf_accessors . . . 88
sf_apa . . . 89
sf_branch . . . 90
sf_branches . . . 91
sf_check . . . 92
sf_checks . . . 93
sf_choice_sets . . . 94
sf_choices . . . 94
sf_component_list . . . 96
sf_conjoint_design . . . 96
sf_construct . . . 98
sf_covariance . . . 99
sf_flagged . . . 99
sf_id . . . 100
sf_identity . . . 101
sf_indirect . . . 101
sf_instrument . . . 102
sf_is_valid . . . 104
sf_item . . . 104
sf_items . . . 106
sf_label . . . 107
sf_meta . . . 108
sf_missing_columns . . . 108
sf_model . . . 109
sf_models . . . 110
sf_object . . . 111
sf_path . . . 111
sf_plan . . . 112
sf_plan<- . . . 113
sf_problems . . . 113
sf_reference . . . 114
sf_report_accessors . . . 116
sf_scale . . . 116
sf_scales . . . 118
sf_validation_accessors . . . 118
sframe_aggregate_judgements . . . 119

sframe_analysis_qmd . . . 120
sframe_as_data_frame . . . 121
sframe_assemble_pairwise . . . 122
sframe_bibliography . . . 123
sframe_builder_empty_state . . . 124
sframe_builder_state_from_instrument . . . 125
sframe_builder_validate_draft . . . 125
sframe_codebook_items_display . . . 127
sframe_collected_weights . . . 127
sframe_decision_options . . . 128
sframe_dematel_compute . . . 129
sframe_demo . . . 130
sframe_demo_branding . . . 131
sframe_demo_data . . . 132
sframe_demo_qmd . . . 132
sframe_demos . . . 133
sframe_export_labelled . . . 134
sframe_input_types_demo_data . . . 135
sframe_likert_scale_groups . . . 136
sframe_plot_cooccurrence . . . 136
sframe_plot_cooccurrence_network . . . 137
sframe_plot_correlation_matrix . . . 138
sframe_plot_decision_ranking . . . 139
sframe_plot_dematel_influence . . . 140
sframe_plot_descriptives . . . 141
sframe_plot_efa_loadings . . . 142
sframe_plot_efa_scree . . . 143
sframe_plot_group_comparison . . . 143
sframe_plot_likert_matrix . . . 144
sframe_plot_likert_scale . . . 145
sframe_plot_missingness . . . 146
sframe_plot_ngram_frequency . . . 147
sframe_plot_paired_comparison . . . 148
sframe_plot_quality . . . 149
sframe_plot_regression_diagnostics . . . 149
sframe_plot_reliability . . . 150
sframe_plot_sentiment . . . 151
sframe_plot_term_frequency . . . 152
sframe_plot_topics . . . 153
sframe_plot_validity . . . 154
sframe_plot_variable_distribution . . . 154
sframe_plots . . . 155
sframe_rated_matrix . . . 157
sframe_references . . . 158
sframe_reports . . . 158
sframe_result_supplement . . . 159
sframe_save_figures . . . 160
sframe_subset . . . 161

sframe_validation 162
summary.sf_branch 163
summary.sf_check 163
summary.sf_choices 164
summary.sf_item 164
summary.sf_model 165
summary.sf_scale 165
summary.sframe 166
survey_module_server 166
survey_module_ui 168
term_context 169
term_frequency 170
theme_surveyframe 171
validate_model 172
validate_sframe 173
validity_report 174
write_sframe 175

Index **177**

`surveyframe-package` *surveyframe: Survey Instrument Workflows for R*

Description

surveyframe defines a survey instrument as a first-class R object and supports a complete workflow from questionnaire design through data collection, quality checking, scoring, psychometric diagnostics, and reproducible reporting. The package covers static HTML survey export, an embeddable Shiny survey module, an interactive response dashboard, a role-based analysis planner with pre-declared research questions, common survey statistics with small-sample alternatives (Hodges-Lehmann, pseudomedian, exact odds-ratio, and Firth logistic regression), multi- criteria decision analysis (AHP, ANP, DEMATEL, VIKOR, MOORA, SMART, WASPAS, PROMETHEE, ELECTRE, and TOPSIS), and model syntax planning for EFA, CFA, CB-SEM, and PLS-SEM.

Core workflow:

1. **Design** an instrument with `launch_builder()` or `sf_instrument()` and its component constructors: `sf_item()`, `sf_choices()`, `sf_scale()`, `sf_branch()`, `sf_check()`.
2. **Validate and save** with `validate_sframe()` and `write_sframe()`. `validate_sframe()` returns an `sframe_validation` diagnostic rather than the instrument itself. Recover a validated instrument with `as_sframe()`.
3. **Deploy** a Shiny survey with `render_survey()`.
4. **Load responses** with `read_responses()` or `read_sheet_responses()`.
5. **Check quality** with `quality_report()`.
6. **Score and analyse** with `score_scales()`, `descriptives_report()`, `missing_data_report()`, `reliability_report()`, `item_report()`, `efa_report()`, `cfa_syntax()`, and `run_analysis_plan()`, which also runs any decision-analysis blocks in the plan.

7. **Report** with `codebook_report()`, `render_report()`, and `render_results()`.

The instrument object:

The workflow runs on an `sframe` object. It is the single source of truth for item definitions, scale structure, reverse-coding keys, branching rules, check specifications, analysis plans, and optional model specifications. Accessors such as `sf_meta()`, `sf_items()`, `sf_scales()`, `sf_plan()`, and `sf_models()` read its parts without reaching into the object directly.

Some helpers work on plain vectors, for use beside that workflow or on their own: the text helpers such as `term_frequency()`, and the interval helpers `bootstrap_ci()`, `cohens_d_ci()`, `cramers_v_ci()` and `eta_sq_ci()`.

A first session:

`sframe_demos()` lists 22 worked demos, each one instrument, its responses and the results surveyframe produced. `sframe_demo("two_group")` loads one, and `sframe_demo_qmd("two_group")` writes a notebook to edit. `vignette("learn-by-example")` teaches from the same library.

How functions are named:

Three families, which the prefix tells apart.

- `sf_` builds or reads the instrument object model: the constructors `sf_item()` and `sf_scale()`, the accessors `sf_items()` and `sf_plan()`, and the replacement forms such as `sf_plan<-`.
- `sframe_` covers everything the package adds around that object: the plots such as `sframe_plot_reliability()`, the demo library through `sframe_demos()`, the decision helpers, and the builder's own state.
- The workflow verbs carry no prefix, because they name the step a researcher is taking: `validate_sframe()`, `score_scales()`, `run_analysis_plan()`, `render_report()`, and the `_report()` family.

The prefixes group functions; they do not pair them. No name stem appears under both, so there is no `sframe_` twin of an `sf_` function to look for.

File format:

Instruments are stored as UTF-8 JSON files with the `.sframe` extension. Each file includes a SHA-256 integrity hash for reproducibility auditing.

Author(s)

Maintainer: Mohammed Ali Sharafuddin <mohammedali.page@gmail.com> (ORCID)

Authors:

- Mohammed Ali Sharafuddin <mohammedali.page@gmail.com> (ORCID)

See Also

Useful links:

- https://mohammedalisharafuddin.github.io/surveyframe/
- https://github.com/MohammedAliSharafuddin/surveyframe
- Report bugs at https://github.com/MohammedAliSharafuddin/surveyframe/issues

add_model *Add a model specification to an instrument*

Description

Add a model specification to an instrument

Usage

```
add_model(instrument, model, validate = TRUE, replace = TRUE)
```

Arguments

instrument	An sframe object.
model	An sf_model() object.
validate	Logical. Whether to validate the model against the instrument before adding it.
replace	Logical. Whether to replace an existing model with the same ID. Defaults to TRUE.

Value

The updated sframe object.

Examples

```
demo <- sframe_demo_data()
m <- sf_model("cb1", type = "cb_sem",
              constructs = list(sf_construct("sat", items = c("sat_1", "sat_2"))))
instr <- add_model(demo$instrument, m)
length(instr$models)
```

add_references *Add references to an instrument*

Description

Attaches sf_reference() records to an instrument. A reference whose id is already present replaces the old one when replace = TRUE, and is refused otherwise, which keeps a second source from silently taking over a citation key. cited_by targets of a replaced reference are merged.

Usage

```
add_references(instrument, ..., replace = FALSE)
```

Arguments

instrument	An sframe object.
...	sf_reference objects, or lists of them such as read_bibtex() returns.
replace	Logical. Replace a reference with the same id.

Details

Adding references changes the file hash, as any edit does, and leaves the content hash that amend_sframe() records unchanged: citing a source is disclosure, outside the design.

Value

The instrument, with the references added.

Examples

```
instr <- sf_instrument("Demo", components = list(
  sf_item("q1", "How satisfied are you?", "likert", choice_set = "agree5"),
  sf_choices("agree5", 1:5, c("SD", "D", "N", "A", "SA"))
))
instr <- add_references(instr, sf_reference(
  "likert_1932", "A technique for the measurement of attitudes",
  author = "Likert, R.", year = 1932, container = "Archives of Psychology",
  volume = "22", issue = "140", pages = "1-55", cited_by = "item:q1"
))
length(sframe_references(instr))
```

---

amend_sframe *Record a disclosed amendment to an instrument*

Description

Appends a structured, disclosed-revision entry to an instrument's amendment log, comparing previous against instrument to record what changed and why: a data-entry correction, bot-response removal, or a documented model respecification. The record is kept inside the file, beside the content it explains.

Usage

```
amend_sframe(
  previous,
  instrument,
  reason_code,
  reason_text,
  tier = NULL,
  author = NULL,
  deviation_report = NULL,
  second_signoff = NULL
)
```

Arguments

previous	An sframe object: the instrument's state before this amendment. Read from a file, it must still match that file, or its last recorded amendment, so the log starts from a state a file held. An edit made to it after reading is refused, and so is one read with verify = FALSE that failed its integrity check.
instrument	An sframe object: the instrument's state after the change this call discloses.
reason_code	One of "data_correction", "bot_removal", "model_respecification", "instrument_revision", "other".
reason_text	Character. A free-text explanation. Required and must be non-empty regardless of reason_code.
tier	"pipeline" or "design". When NULL (the default), inferred from reason_code: data_correction and bot_removal default to "pipeline", and everything else to "design". A change to the analysis plan, a model or a conjoint design is always "design", and asking for "pipeline" on one is an error.
author	Character or NULL. Who made the change.
deviation_report	Character or NULL. Required when tier is "design": what changed in the research question, method, or model, and why. Ignored (may be NULL) for "pipeline" amendments.
second_signoff	Character or NULL. A second reviewer's name or identifier (an ethics board reference, a co-author). When omitted, the entry records signoff = "none".

Details

write_sframe() refuses to write an instrument read from a file whose content has changed with no amendment recorded, one that changed after its last amendment, and one whose amendment log was shortened or reordered. read_sframe() refuses a file edited on disk without its hash being recomputed. These checks are local and the hashes are unsigned: someone who rewrites both a file and its log, and recomputes the hashes, is not detected, and nothing here establishes who made a change or when.

Amendments come in two tiers. A "pipeline" amendment (data corrections, bot removal) needs only a reason. A "design" amendment also requires a deviation_report describing what changed in the research question, method or model, and why. The tier follows the change itself: an amendment that changes the analysis plan, a model or a conjoint design is always design tier, whatever reason_code or tier says. second_signoff is optional. When omitted, the entry records signoff = "none", so the absence of a named reviewer is visible. The tier, report and signoff are what the author records. None of them is independent approval.

previous_hash and new_hash on each entry are a content fingerprint: a SHA-256 over a canonical serialisation of the instrument, with the hash, amendments and references fields excluded, taken after validation, so new_hash is the content that is written. It is distinct from the file's own integrity hash from write_sframe(), which also covers the amendment log. Both identify content. Neither is byte identity.

Value

The amended `sframe` object, with the new entry appended to its amendment log. Call `write_sframe()` to persist it.

Wording on the participant's screens

The welcome, consent and thank-you wording lives in `render`, and both hashes cover `render`, so editing that wording changes the instrument as much as editing an item does. On a published instrument, record it as a pipeline-tier amendment with `reason_code = "other"` and a `reason_text` saying what was reworded, then write the file with `write_sframe()`. This is the intended path for a host that lets researchers edit these screens. A file whose JSON is edited directly fails `read_sframe()`'s hash check.

```
revised <- instrument
revised$render$welcome$consent_text <- "I agree to take part."
revised <- amend_sframe(instrument, revised, "other",
                        "Consent wording shortened", tier = "pipeline")
write_sframe(revised, "study.sframe", overwrite = TRUE)
```

See Also

`amendment_log()`, `write_sframe()`, `read_sframe()`

Examples

```
item  <- sf_item("q1", "How satisfied are you?", type = "text")
instr <- sf_instrument("Demo", components = list(item))
item2 <- sf_item("q1", "How satisfied are you overall?", type = "text")
revised <- sf_instrument("Demo", components = list(item2))
amended <- amend_sframe(
  instr, revised,
  reason_code = "instrument_revision",
  reason_text = "Clarified item wording after a pilot round.",
  deviation_report = "Wording only; no change to the construct measured."
)
nrow(amendment_log(amended))
```

---

amendment_log *Read an instrument's amendment log*

Description

Returns the disclosed-amendment history recorded by `amend_sframe()` as a data frame, one row per amendment in the order they were recorded.

Usage

```
amendment_log(instrument)
```

Arguments

instrument	An sframe object.

Value

A data frame with columns timestamp, reason_code, reason_text, tier, author, deviation_report, signoff, previous_hash, new_hash, and changed_fields (a comma-joined string). Zero rows if the instrument has no recorded amendments. Export with write.csv() for an external audit trail.

See Also

amend_sframe()

Examples

```
item  <- sf_item("q1", "How satisfied are you?", type = "text")
instr <- sf_instrument("Demo", components = list(item))
amendment_log(instr)
```

---

analysis_syntax *The R code behind an analysis result*

Description

Returns the statistical call that produced a result, as R code a reader can copy and run. This is what lets a report show t.test() or cor.test() with the variables and options resolved, rather than only the run_analysis_plan() call that dispatched it.

Usage

```
analysis_syntax(x, which = NULL, data_expr = "scored", header = FALSE)
```

Arguments

x	An sframe_analysis_results object from run_analysis_plan(), or one block's result from it.
which	Character or NULL. One block id, when x holds several.
data_expr	Character. The expression the code should read its data from. Defaults to "scored", the scored frame the header sets up.
header	Logical. Whether to include the lines that load the instrument, read the responses and score the scales. TRUE gives a script that runs on its own.

Details

The code is built from the same resolved specification the runner executed: the variables in the order it resolved them, and the options after defaults were applied. Running it reproduces the statistic, degrees of freedom and p value the package reports, which the package's own tests check by running the generated code and comparing.

Value

A character vector of R code lines, or `NULL` for a method this does not cover. For several blocks, a named list of such vectors.

What it covers

The 2-group, paired, k-group, correlation, regression and categorical families, and descriptives. `sframe_syntax_methods` lists them. A method outside that list returns `NULL`: the model families carry their own syntax already, through `cfa_syntax()` and its neighbours, and for the rest the computation has no single base-R equivalent to show honestly.

See Also

`run_analysis_plan()`, `cfa_syntax()`, `render_report()`

Examples

```
instr <- sf_instrument("Syntax demo", components = list(
  sf_item("score", "Score", type = "numeric"),
  sf_item("arm", "Arm", type = "text")
))
sf_plan(instr) <- list(list(
  id = "RQ1", research_question = "Do the arms differ?",
  family = "inferential", method = "t_test_ind",
  roles = list(group = "arm", outcome = "score")
))

set.seed(1)
responses <- data.frame(
  arm   = rep(c("control", "treatment"), each = 15),
  score = c(rnorm(15, 10), rnorm(15, 12))
)
results <- run_analysis_plan(responses, instr)

# the call behind the number, with the variables and options resolved
cat(analysis_syntax(results, which = "RQ1"), sep = "\n")
```

as.data.frame.sf_choices *Coerce a choice set to a data frame*

Description

Returns the stored values and respondent-facing labels in a choice set.

Usage

```
## S3 method for class 'sf_choices'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A data frame with value and label columns.

See Also

sframe_as_data_frame, sf_choices()

as.data.frame.sframe *Coerce an instrument to its item summary*

Description

Returns one row per item in a surveyframe instrument.

Usage

```
## S3 method for class 'sframe'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A data frame summarising the instrument's items.

See Also

sframe_as_data_frame, sf_items(), codebook_report()

---

as.data.frame.sframe_analysis_results

Summarise analysis-plan results as a data frame

Description

Returns one row per analysis block, including its research question, method, APA summary, and any error.

Usage

```
## S3 method for class 'sframe_analysis_results'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A data frame with one row per analysis block.

See Also

sframe_as_data_frame, run_analysis_plan()

as.data.frame.sframe_assumption_report

Coerce an assumption report to a data frame

Description

Combines the available normality, homogeneity, and regression checks into a long-form summary.

Usage

```
## S3 method for class 'sframe_assumption_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A data frame naming each assumption family, variable, and statistic.

See Also

sframe_as_data_frame, assumption_report()

as.data.frame.sframe_codebook

Extract the item table from a codebook report

Description

Extract the item table from a codebook report

Usage

```
## S3 method for class 'sframe_codebook'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

The codebook report's item table as a data frame.

See Also

sframe_as_data_frame, codebook_report()

---

as.data.frame.sframe_descriptives_report

Extract the descriptives results table

Description

Extract the descriptives results table

Usage

```
## S3 method for class 'sframe_descriptives_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

The primary table from a descriptives report.

See Also

sframe_as_data_frame, descriptives_report()

as.data.frame.sframe_efa_report

Coerce an EFA readiness report to a data frame

Description

Coerce an EFA readiness report to a data frame

Usage

```
## S3 method for class 'sframe_efa_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A one-row data frame containing readiness measures and the suggested number of factors.

See Also

sframe_as_data_frame, efa_report()

as.data.frame.sframe_efa_solution

Extract the loading table from an EFA solution

Description

Extract the loading table from an EFA solution

Usage

```
## S3 method for class 'sframe_efa_solution'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

The solution's long-form factor-loading data frame.

See Also

sframe_as_data_frame, efa_solution()

---

as.data.frame.sframe_item_report

Coerce an item report to a data frame

Description

Stacks item diagnostics from every reported scale into one table.

Usage

```
## S3 method for class 'sframe_item_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A data frame of item-level diagnostics.

See Also

sframe_as_data_frame, item_report()

`as.data.frame.sframe_missing_data_report`
Extract item missingness results

Description

Extract item missingness results

Usage

```
## S3 method for class 'sframe_missing_data_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A surveyframe object.
`row.names`	Passed to `base::as.data.frame()` by the methods that build a frame. Ignored by the methods that return a stored table.
`optional`	Passed to `base::as.data.frame()`.
`...`	Ignored. Present for S3 consistency.

Value

The item-level missingness table from a missing-data report.

See Also

sframe_as_data_frame, `missing_data_report()`

`as.data.frame.sframe_quality_report`
Coerce a response-quality report to a data frame

Description

Coerce a response-quality report to a data frame

Usage

```
## S3 method for class 'sframe_quality_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A one-row data frame summarising respondents, items, and quality flags.

See Also

sframe_as_data_frame, quality_report()

---

as.data.frame.sframe_reliability_report

Coerce a reliability report to a data frame

Description

Returns one row per scale with alpha and omega estimates.

Usage

```
## S3 method for class 'sframe_reliability_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A data frame of scale reliability statistics.

See Also

sframe_as_data_frame, reliability_report()

as.data.frame.sframe_sample_size_plan

Coerce a sample-size plan to a data frame

Description

Coerce a sample-size plan to a data frame

Usage

```
## S3 method for class 'sframe_sample_size_plan'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A surveyframe object.
`row.names`	Passed to `base::as.data.frame()` by the methods that build a frame. Ignored by the methods that return a stored table.
`optional`	Passed to `base::as.data.frame()`.
`...`	Ignored. Present for S3 consistency.

Value

A one-row data frame containing the analysis type, estimated sample size, alpha, and power.

See Also

sframe_as_data_frame, `sample_size_plan()`

as.data.frame.sframe_sensitivity

Extract a sensitivity-analysis results table

Description

Extract a sensitivity-analysis results table

Usage

```
## S3 method for class 'sframe_sensitivity'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

The primary results table from a sensitivity analysis.

See Also

sframe_as_data_frame

---

as.data.frame.sframe_validation

Coerce validation problems to a data frame

Description

Coerce validation problems to a data frame

Usage

```
## S3 method for class 'sframe_validation'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	An sframe_validation object.
row.names	Passed to base::as.data.frame().
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

A data frame with one row per validation problem and columns check and problem.

See Also

sframe_validation, sf_problems()

as.data.frame.sframe_validity_report *Extract reliability evidence from a validity report*

Description

Extract reliability evidence from a validity report

Usage

```
## S3 method for class 'sframe_validity_report'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A surveyframe object.
row.names	Passed to base::as.data.frame() by the methods that build a frame. Ignored by the methods that return a stored table.
optional	Passed to base::as.data.frame().
...	Ignored. Present for S3 consistency.

Value

The reliability table stored in a validity report.

See Also

sframe_as_data_frame, validity_report()

as_sframe *Coerce to an instrument*

Description

Recovers the sframe instrument from a validation diagnostic. This is the migration path for code that used the strict = TRUE return of validate_sframe() as an instrument, which it no longer is.

Usage

```
as_sframe(x, ...)
```

Arguments

x	An sframe_validation object or an sframe.
...	Passed to methods.

Value

An `sframe` object. When the validation passed, its `meta$validated` is `TRUE`.

See Also

`validate_sframe()`, sframe_validation

Examples

```
item  <- sf_item("q1", "How satisfied are you?", type = "text")
instr <- sf_instrument("Demo", components = list(item))

validated <- as_sframe(validate_sframe(instr, strict = TRUE))
isTRUE(sf_meta(validated)$validated)
```

`assumption_report` *Assumption-check report*

Description

Performs common assumption checks for survey analyses using base R where possible: Shapiro-Wilk tests, skewness/kurtosis screening, Levene and Brown-Forsythe tests, regression residual checks, VIF, Cook's distance, expected-count checks, and sparse-cell warnings.

Usage

```
assumption_report(
  data,
  variables = NULL,
  group = NULL,
  outcome = NULL,
  predictors = NULL,
  table_vars = NULL
)
```

Arguments

`data`	A data.frame.
`variables`	Numeric variables for normality screening.
`group`	Optional grouping variable for Levene/Brown-Forsythe tests.
`outcome`	Optional regression outcome.
`predictors`	Optional regression predictors.
`table_vars`	Optional two categorical variables for expected-count checks.

Value

An object of class `sframe_assumption_report`.

Examples

```
demo <- sframe_demo_data()
ar <- assumption_report(demo$responses, variables = c("sat_1", "sat_2"),
                        group = "visit_type")
print(ar)
```

`bootstrap_ci` *Percentile bootstrap confidence interval for a statistic*

Description

Resamples `x` with replacement `R` times, applies `FUN` to each resample, and returns the percentile interval of the resampled statistics together with the observed value.

Usage

```
bootstrap_ci(x, FUN = stats::median, R = 2000, conf.level = 0.95, seed = NULL)
```

Arguments

`x`	A numeric vector.
`FUN`	A function of one vector returning a single number. Defaults to `stats::median()`.
`R`	Integer. Number of bootstrap resamples. Defaults to 2000.
`conf.level`	Confidence level. Defaults to 0.95.
`seed`	Integer or NULL. When supplied, sets the random seed so the interval is reproducible.

Value

A named numeric vector: `estimate`, `lower`, `upper`, with attributes `resamples`, `valid_resamples` and, when the interval is withheld, `reason`. The bounds are `NA` when fewer than 90% of resamples give a value, or when every resample gives the same value, since neither leaves a sampling distribution to read. The bounds are `NA` when `x` has fewer than 3 finite values.

See Also

`cohens_d_ci()`, `cramers_v_ci()`, `eta_sq_ci()`

Examples

```
bootstrap_ci(mtcars$mpg, seed = 42)
bootstrap_ci(mtcars$mpg, FUN = mean, conf.level = 0.90, seed = 42)
```

`cfa_lavaan_syntax` *Generate lavaan CFA syntax from an instrument or a declared model*

Description

The general entry point for CFA syntax. It takes an instrument, and derives the constructs from its scales, or an `sf_model()` declaring constructs that cut across them, and it accepts correlated residuals and latent covariances.

Usage

```
cfa_lavaan_syntax(
  instrument = NULL,
  model = NULL,
  scales = NULL,
  ordered = FALSE,
  std_lv = TRUE,
  residual_covariances = NULL,
  latent_covariances = TRUE
)
```

Arguments

`instrument`	Optional `sframe` object used to derive constructs from scales when `model` is not supplied.
`model`	Optional `sf_model()` object.
`scales`	Optional scale IDs when deriving a model from an instrument.
`ordered`	Logical. Whether to add an ordered-item note.
`std_lv`	Logical. Whether to add a `std.lv = TRUE` note.
`residual_covariances`	Optional list of `sf_covariance()` objects for correlated residuals.
`latent_covariances`	Logical. Whether to include model-level latent covariances supplied in `model`.

Details

`cfa_syntax()` is the instrument-only convenience wrapper, kept for the scripts that call it. Start here when a model is declared, when residual covariances are needed, or when the constructs differ from the scales.

Each generator emits the language its engine reads, so they stay separate: this one and `sem_lavaan_syntax()` write lavaan, `seminr_syntax()` writes seminr, and `efa_syntax()` writes an exploratory plan.

Value

A lavaan syntax string.

Examples

```
demo <- sframe_demo_data()
syntax <- cfa_lavaan_syntax(demo$instrument,
                            scales = c("satisfaction", "behavioural_intention"))
cat(syntax)
```

`cfa_syntax` *Generate lavaan CFA syntax from an instrument object*

Description

Produces a character string of `lavaan` model syntax derived from the scale structure in the instrument. The syntax can be passed directly to `lavaan::cfa()`. Reverse-coded items are noted in a comment but are not transformed in the syntax. Recoding should be applied to the data before fitting the model.

Usage

```
cfa_syntax(instrument, scales = NULL, std_lv = TRUE)
```

Arguments

`instrument`	An `sframe` object.
`scales`	Character vector or NULL. A subset of scale IDs to include. When NULL, all scales are included.
`std_lv`	Logical. Whether to include the `std.lv = TRUE` argument note in the output comment header. Defaults to `TRUE`.

Value

A character string of `lavaan` CFA model syntax.

Which of the two to use

This is the instrument-only convenience wrapper: one call, constructs taken from the instrument's scales. `cfa_lavaan_syntax()` is the general entry point, and takes those same arguments plus a declared `sf_model()`, correlated residuals and latent covariances. Reach for it where the constructs differ from the scales. This wrapper stays supported, so a script calling it keeps working.

See Also

`cfa_lavaan_syntax()`, `efa_report()`, `reliability_report()`

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
i1    <- sf_item("sat_1", "Item 1", type = "likert",
                 choice_set = "ag5", scale_id = "sat")
i2    <- sf_item("sat_2", "Item 2", type = "likert",
                 choice_set = "ag5", scale_id = "sat")
i3    <- sf_item("sat_3", "Item 3 (reverse)", type = "likert",
                 choice_set = "ag5", scale_id = "sat", reverse = TRUE)
scale <- sf_scale("sat", "Satisfaction",
                  items = c("sat_1", "sat_2", "sat_3"))
instr <- sf_instrument("Demo Survey", components = list(cs, i1, i2, i3, scale))

syntax <- cfa_syntax(instr)
cat(syntax)

## Not run:
# lavaan is not installed by default. Install it before fitting.
demo   <- sframe_demo_data()
scored <- score_scales(demo$responses, demo$instrument)
fit    <- lavaan::cfa(syntax, data = scored, std.lv = TRUE)
summary(fit, fit.measures = TRUE)

## End(Not run)
```

`clean_text_responses` *Clean open-ended text responses for analysis*

Description

Extracts one text/textarea item's responses from a response data frame and applies light, configurable cleaning: lower-casing, punctuation removal, and optional number stripping. Blank and missing responses are dropped rather than kept as empty strings, since they carry no term-frequency signal and would otherwise inflate downstream response counts.

Usage

```
clean_text_responses(
  data,
  item_id,
  lowercase = TRUE,
  remove_punct = TRUE,
  strip_numbers = FALSE,
  instrument = NULL
)
```

Arguments

data	A data.frame of responses.
item_id	Character. The text/textarea item's column name.
lowercase	Logical. Lower-case the text. Default TRUE.
remove_punct	Logical. Strip punctuation. Default TRUE.
strip_numbers	Logical. Strip digits. Default FALSE.
instrument	Optional sframe instrument. When supplied, item_id is validated as a "text" or "textarea" item before cleaning.

Value

A character vector of cleaned responses, with an integer "respondent" attribute giving each entry's original row index in data, so quotes extracted later can cite a respondent.

Examples

```
demo <- sframe_demo_data()
cleaned <- clean_text_responses(demo$responses, "comments")
head(cleaned)
attr(cleaned, "respondent")[1:5]
```

---

codebook_report *Generate a survey codebook from an instrument object*

Description

Produces a structured codebook listing all items, their types, choice sets, scale membership, and reverse-coding status. The codebook can be rendered as HTML or Markdown.

Usage

```
codebook_report(instrument, format = c("html", "md"))
```

Arguments

instrument	An sframe object.
format	Character. Output format. Either "html" or "md". With "md", the codebook also carries itself as Markdown text, in markdown, and print() writes that text.

Value

An object of class sframe_codebook, a list with elements instrument_meta, items_table, choices_table, scales_table, plan_table, models_table and format, and with format = "md", markdown: one character string holding the whole codebook as Markdown, with a pipe table per component. Save it with writeLines(cb$markdown, "codebook.md"). With "html", print() shows a compact summary, and render_report() includes the codebook in a full report.

See Also

render_report()

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
i1    <- sf_item("sat_1", "Item 1", type = "likert",
                 choice_set = "ag5", scale_id = "sat")
i2    <- sf_item("sat_2", "Item 2", type = "likert",
                 choice_set = "ag5", scale_id = "sat")
scale <- sf_scale("sat", "Satisfaction", items = c("sat_1", "sat_2"))
instr <- sf_instrument("Demo Survey", components = list(cs, i1, i2, scale))

cb <- codebook_report(instr)
print(cb)
md <- codebook_report(instr, format = "md")
cat(substr(md$markdown, 1, 200))
nrow(sf_items(cb))
nrow(sf_scales(cb))
```

---

cohens_d_ci *Bootstrap confidence interval for Cohen's d*

Description

Percentile bootstrap for the standardised mean difference between two independent groups. The difference is signed, the mean of x minus the mean of y, so the interval can include zero. Each resample draws within each group, preserving the group sizes.

Usage

```
cohens_d_ci(x, y, R = 2000, conf.level = 0.95, seed = NULL)
```

Arguments

x, y	Numeric vectors, one per group.
R	Integer. Number of bootstrap resamples. Defaults to 2000.
conf.level	Confidence level. Defaults to 0.95.
seed	Integer or NULL. When supplied, sets the random seed.

Value

A named numeric vector: estimate, lower, upper, with attributes resamples, valid_resamples and, when the interval is withheld, reason. The bounds are NA when fewer than 90% of resamples give a value, or when every resample gives the same value, since neither leaves a sampling distribution to read. The bounds are NA when either group has fewer than 3 finite values.

See Also

bootstrap_ci()

Examples

```
cohens_d_ci(mtcars$mpg[mtcars$am == 1], mtcars$mpg[mtcars$am == 0],
            seed = 42)
```

`cramers_v_ci` *Bootstrap confidence interval for Cramer's V*

Description

Percentile bootstrap for the association strength in a contingency table. The table is expanded back to individual observations, which are resampled jointly. For a 2 by 2 table the statistic equals phi.

Usage

```
cramers_v_ci(tab, R = 2000, conf.level = 0.95, seed = NULL)
```

Arguments

`tab`	A contingency table (from `table()`) or a matrix of counts.
`R`	Integer. Number of bootstrap resamples. Defaults to 2000.
`conf.level`	Confidence level. Defaults to 0.95.
`seed`	Integer or NULL. When supplied, sets the random seed.

Value

A named numeric vector: `estimate`, `lower`, `upper`, with attributes `resamples`, `valid_resamples` and, when the interval is withheld, `reason`. The bounds are `NA` when fewer than 90% of resamples give a value, or when every resample gives the same value, since neither leaves a sampling distribution to read. The bounds are `NA` when the table holds fewer than 3 observations.

See Also

bootstrap_ci()

Examples

```
cramers_v_ci(table(mtcars$am, mtcars$cyl), seed = 42)
```

`descriptives_report` *Descriptive statistics report*

Description

Computes survey descriptives for numeric, Likert, and scale-score columns, including missingness, mean, standard deviation, median, IQR, range, skewness, kurtosis, standard error, and confidence intervals.

Usage

```
descriptives_report(
  data,
  variables = NULL,
  split_by = NULL,
  conf_level = 0.95,
  weights = NULL
)
```

Arguments

`data`	A data.frame of responses.
`variables`	Character vector of variables. When `NULL`, numeric-like columns are used.
`split_by`	Optional grouping variable.
`conf_level`	Confidence level for the mean interval.
`weights`	Optional case-weight column.

Value

An object of class `sframe_descriptives_report`.

Examples

```
demo <- sframe_demo_data()
dr <- descriptives_report(demo$responses, variables = c("sat_1", "sat_2"),
                          split_by = "visit_type")
dr$table
```

`efa_report` *Prepare a survey instrument for exploratory factor analysis*

Description

Reports KMO sampling adequacy, Bartlett's test of sphericity, and a parallel analysis scree plot to inform factor number selection. The suggested number of factors from parallel analysis is returned in `$suggested_nfactors`. The report prepares the researcher to estimate an EFA solution with a separate package such as `psych` or `lavaan`.

Usage

```
efa_report(
  data,
  instrument,
  scales = NULL,
  nfactors = NULL,
  rotation = "oblimin"
)
```

Arguments

`data`	A `tibble` or `data.frame` of responses.
`instrument`	An `sframe` object.
`scales`	Character vector or NULL. Scale IDs whose items to include. When NULL, all scale items are pooled.
`nfactors`	Integer or NULL. Suggested number of factors to highlight on the scree plot. When NULL, the parallel analysis recommendation is used.
`rotation`	Character. The rotation method to display in the diagnostic notes. Does not affect the diagnostics themselves. Defaults to `"oblimin"`.

Value

An object of class `sframe_efa_report` with elements `kmo`, `bartlett`, `parallel`, and `suggested_nfactors`.

Fitting the solution

This report says whether the data suit a factor analysis and how many factors to extract. `efa_solution()` then fits that solution and returns the loadings, so the whole route stays in surveyframe. Take the result to another package when you want a method surveyframe leaves out.

See Also

`efa_solution()`, `reliability_report()`, `cfa_syntax()`

Examples

```
if (requireNamespace("psych", quietly = TRUE)) {
  demo <- sframe_demo_data()
  er <- efa_report(demo$responses, demo$instrument)
  print(er)
}
```

`efa_solution` *Estimate an exploratory factor solution*

Description

Runs `psych::fa()` on selected item columns and returns loadings, communalities, uniqueness, variance summaries, and simple item retention flags. The `psych` package is optional and is only required when this function is called.

Usage

```
efa_solution(
  data,
  instrument,
  items = NULL,
  scales = NULL,
  nfactors = 1L,
  extraction = c("minres", "pa", "ml"),
  rotation = c("oblimin", "promax", "varimax"),
  min_loading = 0.3,
  cross_loading = 0.3
)
```

Arguments

`data`	A data.frame of responses.
`instrument`	An `sframe` object.
`items`	Character vector of item IDs. When `NULL`, scale items are used.
`scales`	Optional scale IDs used to select item columns.
`nfactors`	Number of factors.
`extraction`	Extraction method passed to `psych::fa()`.
`rotation`	Rotation method passed to `psych::fa()`.
`min_loading`	Minimum salient loading.
`cross_loading`	Maximum secondary loading before a warning is raised.

Value

An object of class `sframe_efa_solution`. Alongside the psych objects it carries three tidy data frames ready for plotting and reporting: `loadings_long` (item_id, factor, loading), `communalities_table` (item_id, communality, uniqueness), and `variance_table` (factor, ss_loadings, proportion_var, cumulative_var).

Examples

```
if (requireNamespace("psych", quietly = TRUE)) {
  demo <- sframe_demo_data()
  fit <- efa_solution(demo$responses, demo$instrument,
                      scales = "service_quality", nfactors = 1)
  fit$loadings
}
```

`efa_syntax` *Generate EFA planning syntax*

Description

Generate EFA planning syntax

Usage

```
efa_syntax(
  items,
  nfactors = 1L,
  extraction = c("minres", "pa", "ml"),
  rotation = c("oblimin", "promax", "varimax"),
  data_name = "data"
)
```

Arguments

`items`	Character vector of item IDs.
`nfactors`	Number of factors.
`extraction`	Extraction method.
`rotation`	Rotation method.
`data_name`	Name of the data object in generated R code.

Value

A character string with R syntax.

Examples

```
syntax <- efa_syntax(c("sq_1", "sq_2", "sq_3"), nfactors = 1)
cat(syntax)
```

`eta_sq_ci` *Bootstrap confidence interval for eta squared*

Description

Percentile bootstrap for the proportion of variance in `outcome` explained by `group`, resampling observations jointly so the group structure travels with each resample.

Usage

```
eta_sq_ci(outcome, group, R = 2000, conf.level = 0.95, seed = NULL)
```

Arguments

`outcome`	A numeric vector.
`group`	A grouping vector of the same length.
`R`	Integer. Number of bootstrap resamples. Defaults to 2000.
`conf.level`	Confidence level. Defaults to 0.95.
`seed`	Integer or NULL. When supplied, sets the random seed.

Value

A named numeric vector: `estimate`, `lower`, `upper`, with attributes `resamples`, `valid_resamples` and, when the interval is withheld, `reason`. The bounds are `NA` when fewer than 90% of resamples give a value, or when every resample gives the same value, since neither leaves a sampling distribution to read. The bounds are `NA` with fewer than 3 complete observations or fewer than 2 groups.

See Also

`bootstrap_ci()`

Examples

```
eta_sq_ci(mtcars$mpg, mtcars$cyl, seed = 42)
```

`export_google_sheet` *Export a survey instrument to Google Sheets collection format*

Description

Generates a Google Apps Script file that, when run in a Google Sheet, creates a response collection endpoint for a survey instrument. The builder can store the deployed Apps Script URL in survey metadata, and the same sheet can be read back with `read_sheet_responses()`.

Usage

```
export_google_sheet(instrument, sheet_url, output_dir = ".")
```

Arguments

`instrument`	An `sframe` object.
`sheet_url`	Character. The URL of an existing Google Sheet. The generated script records it as a comment and a constant, so the file identifies the sheet it was written for. The collector writes through the active spreadsheet it is bound to, so the sheet needs no change to its sharing settings.
`output_dir`	Character. Directory to write the Apps Script file. Defaults to the current working directory.

Value

The path to the generated `.gs` Apps Script file, invisibly.

Changing the instrument mid-collection

Regenerating and redeploying the collector after an instrument change is safe, and it is what a pilot study normally does after a face-validity pass. The collector maps every response to the sheet's own header by name, so a column that already exists never moves and rows collected before the change stay valid. An item added to the instrument gets a new column at the right-hand end of the sheet, and rows collected before it existed are left blank in that column.

Before surveyframe 0.4.1 this was not true. The header was written once, at sheet creation, and rows were built positionally, so an item added mid-collection shifted every value from the insertion point onward into the wrong column, silently. If you collected responses through a redeployed collector generated by 0.4.0 or earlier, check the sheet's header against the instrument before analysing it.

Who can reach the collected responses

Keep the sheet private. The collector is an Apps Script bound to the sheet and writes through `SpreadsheetApp.getActiveSpreadsheet()`, so it runs with the access of whoever deployed it. No sharing change is needed for collection to work.

Step 5 of the generated setup sets the web app's "Who has access" to "Anyone". That setting belongs to the web app endpoint, which accepts submissions. It grants no access to the sheet itself.

Sharing the sheet so that any link holder can edit would expose every collected response, personal data included, to anyone holding the URL, and would let them alter or delete it. Before surveyframe 0.4.2 this help advised exactly that. If you followed it, review the sharing settings on any sheet you collected into.

`read_sheet_responses()` reads through `googlesheets4`, which authenticates as you, so a private sheet is readable with no sharing change.

See Also

`read_sheet_responses()`, `read_responses()`, `write_sframe()`

Examples

```
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
script <- export_google_sheet(
  instr,
  sheet_url = "https://docs.google.com/spreadsheets/d/demo",
  output_dir = tempdir()
)
file.exists(script)
```

---

export_static_survey *Export a self-contained static HTML survey*

Description

Generates a single HTML file that presents the survey instrument in a browser by itself, offline, with the Shiny server left out. Every item type, branching, required-item checks and multi-page navigation run in the browser's own JavaScript.

Usage

```
export_static_survey(
  instrument,
  output_path = NULL,
  open = interactive(),
  endpoint_url = NULL,
  overwrite = FALSE,
  preview = FALSE
)
```

Arguments

instrument	An sframe object.
output_path	Character. File path for the output HTML. When NULL, a <survey_title>.html file is written in tempdir().
open	Logical. If TRUE (default) and the session is interactive, the file is opened in the default browser after writing.
endpoint_url	Character or NULL. A URL to which responses are POSTed as JSON on submission. When NULL, CSV download is the only collection mechanism.
overwrite	Logical. Whether to overwrite an existing file at output_path. Defaults to FALSE.
preview	Logical. TRUE exports a survey that collects nothing: the collector endpoint and the completion redirect are both removed, and the thank-you screen says the response went nowhere, above the study's own message as respondents will read it. This is what SurveyStudio's preview uses, which keeps a test answer away from a live study's collector. Supplying endpoint_url alongside it is an error. Defaults to FALSE.

Details

When output_path is NULL, the file is written to tempdir(). Supply an explicit output_path for any production export that should be kept.

Value

The output path, invisibly.

How a response reaches you

On submission the survey builds a one-row CSV in the browser's memory and shows the thank-you screen. When endpoint_url is supplied, it also sends the response as a POST request to that URL, for example a Google Apps Script web app. The browser reads back none of the reply to that request, so the survey treats it as sent and the respondent sees the thank-you screen either way.

The thank-you screen offers the CSV as a download button, which the respondent chooses to use. It appears when the instrument's thank-you settings ask for it, and whenever there is no endpoint_url, where that file is the only copy of the response.

Plan for both parts. Test the endpoint with a pilot submission and confirm the row arrives before collecting, and treat the download as a route a respondent may decline.

For a host page: confirmed delivery, restarts and drafts

While a response is being sent, the survey shows a sending screen. A host that can confirm delivery, such as a platform embedding the survey, defines window.sframeSendResponse(row) in the page before the survey script runs. The survey then hands each response to that function in place of its own POST. It returns a Promise: resolving to true, or to {confirmed: true}, shows the response as recorded. Resolving to anything else shows it as sent and unconfirmed, and a rejection shows the failure screen with a retry.

The thank-you screen states delivery itself, one line for each outcome: recorded, sent but unconfirmed, not sent, or failed. Write `render$thankyou$message` as thanks, a debrief or contact details, and do not state delivery in it. It appears below the status line in every case except a failed send, and defaults to "Thank you for participating."

Two settings in the instrument's `render` block suit a host too. `render$thankyou$show_restart = FALSE` hides "Submit another response". `render$save_draft = TRUE` keeps the answers in the browser's local storage as the respondent goes, under a key drawn from the instrument's content, so a respondent who closes the tab is offered to resume on return. The draft is cleared once the response is submitted, and kept when delivery fails.

The exported file works offline. It can be hosted on GitHub Pages, Netlify, any static file server, or e-mailed as an attachment for opening directly from disk.

See Also

`launch_studio()`, `launch_builder()`, `render_survey()`

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
i1    <- sf_item("sat_1", "Overall I am satisfied with the service.",
                 type = "likert", choice_set = "ag5", required = TRUE)
i2    <- sf_item("comments", "Any additional comments?", type = "textarea")
instr <- sf_instrument("Customer Satisfaction Survey",
                       components = list(cs, i1, i2))

# Write to a temp file without opening the browser
out <- export_static_survey(instr,
                            output_path = file.path(tempdir(), "sat.html"),
                            open = FALSE)
file.exists(out)


# Write to a temp file and open in the default browser
export_static_survey(instr,
                     output_path = file.path(tempdir(), "sat_browser.html"),
                     overwrite = TRUE)

# Write with a Google Apps Script endpoint for server-side collection
export_static_survey(
  instr,
  output_path  = file.path(tempdir(), "sat_endpoint.html"),
  endpoint_url = "https://script.google.com/macros/s/XXXXX/exec",
  open         = FALSE,
  overwrite    = TRUE
)
```

`extract_quotes` *Extract representative quotes for each STM topic*

Description

Takes the result of `sframe_run_stm_topics()` (not a raw `stm` model object) and, for each topic, pulls the top `n_quotes` documents by topic-document probability using `stm::findThoughts()`, then maps each one back to its original respondent row via the mapping `sframe_run_stm_topics()` stored on `result$fit`.

Usage

```
extract_quotes(model, text, n_quotes = 3L)
```

Arguments

`model`	A `stm_topics` result list from `sframe_run_stm_topics()` (i.e. `result`, not `result$fit` and not the raw `stm` object).
`text`	The response vector the topic model was fit on: either the raw vector (one entry per original data row, indexed 1:1 by row number) or the `clean_text_responses()`-cleaned vector, which drops blank/missing rows and is therefore *shorter*, its positions no longer equal original row numbers once any earlier row was dropped. Both forms work correctly: when `text` carries the `respondent` attribute `clean_text_responses()` sets, that mapping is used to find each quote's real position; otherwise `text` is assumed to be the raw, 1:1-indexed vector.
`n_quotes`	Integer. Quotes to return per topic. Default `3L`.

Value

A data.frame with columns `topic`, `rank`, `respondent` (the original row index in the data the model's `text` argument came from, not a document-matrix or corpus row index), and `quote`.

Examples

```
if (requireNamespace("stm", quietly = TRUE) &&
    requireNamespace("tidytext", quietly = TRUE)) {
  demo <- sframe_demo_data()
  sf_plan(demo$instrument) <- list(list(
    id = "RQ1", research_question = "What themes appear in the comments?",
    family = "text_analysis", method = "stm_topics",
    roles = list(item = "comments"), options = list(k = 3, seed = 42)
  ))
  res <- run_analysis_plan(demo$responses, demo$instrument)
  quotes <- extract_quotes(res$RQ1, demo$responses$comments, n_quotes = 2)
  quotes
}
```

format.sf_branch *Format an sf_branch object as a string*

Description

Format an sf_branch object as a string

Usage

```
## S3 method for class 'sf_branch'
format(x, ...)
```

Arguments

x	An object of class sf_branch.
...	Ignored. Present for S3 consistency.

Value

A single character string.

format.sf_check *Format an sf_check object as a string*

Description

Format an sf_check object as a string

Usage

```
## S3 method for class 'sf_check'
format(x, ...)
```

Arguments

x	An object of class sf_check.
...	Ignored. Present for S3 consistency.

Value

A single character string.

format.sf_choices *Format an sf_choices object as a string*

Description

Format an sf_choices object as a string

Usage

```
## S3 method for class 'sf_choices'
format(x, ...)
```

Arguments

x	An object of class sf_choices.
...	Ignored. Present for S3 consistency.

Value

A single character string.

format.sf_item *Format an sf_item object as a string*

Description

Format an sf_item object as a string

Usage

```
## S3 method for class 'sf_item'
format(x, ...)
```

Arguments

x	An object of class sf_item.
...	Ignored. Present for S3 consistency.

Value

A single character string.

format.sf_model *Format an sf_model object as a string*

Description

Format an sf_model object as a string

Usage

```
## S3 method for class 'sf_model'
format(x, ...)
```

Arguments

x	An object of class sf_model.
...	Ignored. Present for S3 consistency.

Value

A single character string.

format.sf_scale *Format an sf_scale object as a string*

Description

Format an sf_scale object as a string

Usage

```
## S3 method for class 'sf_scale'
format(x, ...)
```

Arguments

x	An object of class sf_scale.
...	Ignored. Present for S3 consistency.

Value

A single character string.

format.sframe *Format an sframe instrument object as a string*

Description

Format an sframe instrument object as a string

Usage

```
## S3 method for class 'sframe'
format(x, ...)
```

Arguments

x	An object of class sframe.
...	Ignored. Present for S3 consistency.

Value

A single character string.

item_report *Generate item-level diagnostics*

Description

Produces, for each item within each scale, the item-rest correlation, floor and ceiling proportions, and the item mean and standard deviation.

Usage

```
item_report(data, instrument, scales = NULL)
```

Arguments

data	A tibble or data.frame of responses.
instrument	An sframe object.
scales	Character vector or NULL. A subset of scale IDs to analyse. When NULL (default), all scales are included.

Details

Diagnostics use the scale's scoring orientation, so an item the scale reverse-codes is reversed first, as in `score_scales()` and `reliability_report()`. The item-rest correlation is the correlation between an item and the sum of the scale's other items. It is computed on respondents who answered every item in the scale, the same rows `reliability_report()` uses, and `n_missing` counts the item's own missing values in `data`.

Floor and ceiling are the proportions at the item's declared lowest and highest response, taken from its choice set, slider limits or rating maximum. They are `NA` for an item that declares no bounds, since the sample's own extremes say nothing about a floor or ceiling effect.

Value

An object of class `sframe_item_report`: a named list with one element per scale, each a list holding `scale_id`, `label` and `diagnostics`, a data frame with one row per item and columns `item_id`, `mean`, `sd`, `item_rest_r`, `floor_pct`, `ceiling_pct` and `n_missing`. `as.data.frame()` stacks every scale's diagnostics into one table.

See Also

`reliability_report()`, `sf_scale()`

Examples

```
demo <- sframe_demo_data()
ir <- item_report(demo$responses, demo$instrument)
print(ir)
# one scale's diagnostics
ir[[1]]$diagnostics
# every scale in one table
as.data.frame(ir)
```

---

launch_builder *Launch the surveyframe visual survey builder*

Description

Opens the SurveyBuilder, a self-contained HTML application for visual survey design. The builder runs client-side without an R session or Shiny server. Save instruments as `.sframe` files from the browser and load them into R with `read_sframe()`.

Usage

```
launch_builder(open = TRUE)
```

Arguments

open	Logical. When TRUE (the default), the builder HTML file is opened in the system's default web browser with utils::browseURL(). Set to FALSE to return the file path without opening it, which is useful for automated testing.

Details

The builder includes a three-mode interface.

Build An item editor with a persistent inspector panel, drag-to-reorder, undo/redo, and autosave to browser localStorage.

Preview A layout preview of the welcome, body and thank-you pages, showing wording, order and branding. It renders from the builder's own markup, so answering, required checks and branching are left out. Use export_static_survey() and open the file to test the respondent's path, or SurveyStudio's Preview screen, which exports the real survey with collection switched off.

Analyse A role-based analysis planner with method-specific options, planned outputs, reporting references, and decision rules.

The builder includes a pure-JavaScript SHA-256 fallback for browsers or security policies where crypto.subtle is unavailable on file:// origins. Saved .sframe files can be loaded and validated with read_sframe().

Value

The path to the bundled builder HTML file, invisibly.

See Also

launch_studio(), read_sframe(), run_analysis_plan()

Examples

```
# Retrieve the builder path for inspection without opening the browser
path <- launch_builder(open = FALSE)
file.exists(path)
```

---

launch_builder_demo *Launch SurveyBuilder with the bundled input-types demo preloaded*

Description

Opens a temporary copy of the SurveyBuilder with the bundled input-types instrument already injected into the JavaScript state. The demo questions, scales, and analysis plan are visible immediately, with no manual file-load step required.

Usage

```
launch_builder_demo(open = TRUE)
```

Arguments

open	Logical. When TRUE (the default), the pre-populated builder HTML is opened in the system's default web browser.

Value

Invisibly returns a list with builder_path, demo_file, and responses_path.

Examples

```
demo <- sframe_input_types_demo_data()
nrow(demo$responses)
## Not run:
launch_builder_demo()

## End(Not run)
```

---

launch_dashboard *Launch the interactive response dashboard*

Description

Opens a Shiny dashboard to explore collected response data alongside the instrument definition. Use this interface after response collection for analysis and quality control. Use launch_builder() to design new questionnaires. The dashboard includes five panels:

Usage

```
launch_dashboard(
  instrument = NULL,
  responses = NULL,
  port = NULL,
  host = "127.0.0.1",
  launch.browser = interactive()
)
```

Arguments

instrument	An sframe object. Required. Calling launch_dashboard() with no instrument errors with guidance. Use launch_dashboard_demo() for the bundled demo or launch_studio() to upload interactively.
responses	A data.frame or tibble of survey responses, as produced by read_responses() or read_sheet_responses(). When NULL the dashboard opens with instrument metadata and no response summaries.

`port`	Integer or NULL. TCP port for the Shiny server. When NULL, Shiny selects an available port automatically.
`host`	Character. Host address passed to `shiny::runApp()`. Defaults to `"127.0.0.1"`.
`launch.browser`	Logical. Whether to open the dashboard in the default browser automatically. Defaults to `TRUE` in interactive sessions.

Details

Overview Response count, date range, and instrument metadata.

Items Per-item frequency bar charts, histograms, and tabulated frequency counts for choice-type questions.

Scales Scale score distributions with mean overlay, and a summary table of scale definitions.

Quality Attention check pass rates for each check defined in the instrument.

Raw data Scrollable response table with a CSV download button.

The dashboard is read-only and takes its data from R. It has no upload screen, so pass `instrument` and `responses` directly. To open and upload data interactively, use `launch_studio()`, which includes this same dashboard as its Dashboard tab. For a quick look at bundled demo data, use `launch_dashboard_demo()`.

Value

Called for its side effect. Returns nothing.

See Also

`run_analysis_plan()`, `quality_report()`, `score_scales()`

Examples

```
## Not run:
# For the bundled demo, use launch_dashboard_demo().
# To upload data interactively, use launch_studio().

# Open the dashboard with your own instrument and responses
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
responses <- read_responses(
  system.file("extdata", "tourism_services_responses.csv",
              package = "surveyframe"),
  instr,
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  meta_cols = "started_at"
)
launch_dashboard(instr, responses)

## End(Not run)
```

launch_dashboard_demo *Launch the response dashboard with the bundled input-types demo*

Description

Opens the dashboard with the bundled input-types questionnaire and 120 simulated responses already loaded. The browser is opened automatically by default.

Usage

```
launch_dashboard_demo(port = NULL, host = "127.0.0.1", launch.browser = TRUE)
```

Arguments

port	TCP port for the Shiny server.
host	Host address for the Shiny server.
launch.browser	Whether to open the browser automatically. Defaults to TRUE for this demo helper.

Value

Called for its side effect.

Examples

```
demo <- sframe_input_types_demo_data()
nrow(demo$responses)
## Not run:
launch_dashboard_demo()

## End(Not run)
```

launch_studio *Launch the SurveyStudio interface*

Description

Opens the SurveyStudio Shiny application, the visual interface for working with an instrument that already exists. Its screens open an instrument, record and read amendments, preview the survey, upload responses, review data quality, inspect reliability, work on the analysis plan, read the dashboard, and export.

Usage

```
launch_studio(
  instrument = NULL,
  responses = NULL,
  respondent_id = NULL,
  submitted_at = NULL,
  meta_cols = NULL,
  strict = TRUE,
  screen = "auto",
  port = NULL,
  host = "127.0.0.1",
  launch.browser = interactive()
)
```

Arguments

instrument	An sframe object or NULL.
responses	A data.frame, tibble, CSV file path, or NULL.
respondent_id	Character or NULL. Response ID column when responses is a CSV path.
submitted_at	Character or NULL. Submission time column when responses is a CSV path.
meta_cols	Character vector or NULL. Metadata columns when responses is a CSV path.
strict	Logical. Passed to read_responses() when responses is a CSV path.
screen	The screen to open on. One of "auto", which picks by what you supply, or a screen name: "open", "amendments", "references", "preview", "responses", "quality", "reliability", "analysis", "dashboard" or "export". "data" is accepted for "responses".
port	TCP port for the Shiny server.
host	Host address passed to shiny::runApp().
launch.browser	Whether to open the browser automatically.

Details

Studio reads and analyses an instrument. To author one, question by question, use launch_builder(), and open the result here.

Value

Called for its side effect.

See Also

launch_builder(), launch_dashboard(), read_sframe(), read_responses()

Examples

```
## Not run:
launch_studio()

demo <- sframe_demo_data()
launch_studio(instrument = demo$instrument, launch.browser = FALSE)

launch_studio(
  instrument    = demo$instrument,
  responses     = demo$responses,
  respondent_id = "respondent_id",
  submitted_at  = "submitted_at"
)

## End(Not run)
```

---

launch_studio_demo *Launch SurveyStudio with the bundled input-types demo*

Description

Opens SurveyStudio with the bundled input-types questionnaire and simulated response data already loaded. The browser is opened automatically by default.

Usage

```
launch_studio_demo(
  screen = "preview",
  port = NULL,
  host = "127.0.0.1",
  launch.browser = TRUE
)
```

Arguments

`screen`	Initial studio screen. Defaults to `"preview"` so the demo content is immediately visible.
`port`	TCP port for the Shiny server.
`host`	Host address for the Shiny server.
`launch.browser`	Whether to open the browser automatically. Defaults to `TRUE` for this demo helper.

Value

Called for its side effect.

Examples

```
demo <- sframe_input_types_demo_data()
nrow(demo$responses)
## Not run:
launch_studio_demo()

## End(Not run)
```

`link_git_commit` *Link an instrument to its current Git commit*

Description

Records the current Git commit SHA and subject line for `repo_path`. It is a pointer into Git history, where a reviewer reads what changed and why.

Usage

```
link_git_commit(instrument, repo_path = ".", path = NULL)
```

Arguments

`instrument`	An `sframe` object.
`repo_path`	Character. Path to check for a Git repository. Defaults to the current working directory.
`path`	Character or `NULL`. The instrument's `.sframe` file, relative to `repo_path`. When given, the instrument is compared with the file as committed at HEAD.

Details

Given `path`, the tracked `.sframe` file, it also compares the instrument with that file as committed at HEAD, and sets `verified = TRUE` only when their content hashes match. Without `path`, nothing is compared and `verified` is `FALSE`. A verified link shows the instrument matches a committed file. It does not show who wrote the instrument or when, beyond what the commit itself records.

Git is entirely optional. When `repo_path` is not inside a Git repository, or the `git` executable is not on the `PATH`, this returns a clear, non-error result with `linked = FALSE` rather than aborting – the rest of surveyframe never requires Git.

Value

A list with `linked` (logical), and when `linked` is `TRUE`, `commit` (the full commit SHA), `message` (the commit's subject line), `path`, and `verified` (logical). `reason` explains an unlinked result (`"git not found"` or `"not a git repository"`) or an unverified one.

See Also

`amend_sframe()`, `write_sframe()`

Examples

```
item  <- sf_item("q1", "How satisfied are you?", type = "text")
instr <- sf_instrument("Demo", components = list(item))
link_git_commit(instr, repo_path = tempdir())
```

`missing_data_report` *Missing-data report*

Description

Reports item-wise missingness, respondent-wise missingness, missing-data patterns, listwise and pairwise deletion counts, and scale scoring missing rules. No imputation is performed.

Usage

```
missing_data_report(data, instrument = NULL, variables = NULL)
```

Arguments

`data`	A data.frame of responses.
`instrument`	Optional `sframe` object.
`variables`	Optional response columns. Defaults to instrument item IDs when an instrument is supplied, otherwise all columns.

Value

An object of class `sframe_missing_data_report`.

Examples

```
demo <- sframe_demo_data()
mr <- missing_data_report(demo$responses, demo$instrument)
mr$item_missing
```

`model_json` *Serialise a model specification to JSON*

Description

Serialise a model specification to JSON

Usage

```
model_json(model, pretty = TRUE)
```

Arguments

model	An sf_model() object.
pretty	Logical. Whether to pretty-print the JSON.

Value

A JSON string.

Examples

```
m <- sf_model("cb1", type = "cb_sem",
              constructs = list(sf_construct("sq", items = c("sq_1", "sq_2"))))
cat(model_json(m))
```

---

model_report_template *Create a model reporting template*

Description

Create a model reporting template

Usage

```
model_report_template(model, include_json = TRUE)
```

Arguments

model	An sf_model() object.
include_json	Logical. Whether to include the JSON schema block.

Value

A character string.

Examples

```
m <- sf_model("cb1", type = "cb_sem",
              constructs = list(sf_construct("sat", items = c("sat_1", "sat_2"))))
cat(model_report_template(m, include_json = FALSE))
```

ngram_frequency *N-gram frequency for open-ended text*

Description

Tokenises `text` via the same tokeniser as `term_frequency()` (whitespace splitting, lower-casing, punctuation stripping, and stop-word removal), then slides a window of `n` tokens across each response's token vector and counts how often each resulting n-gram occurs. `n = 2` (the default) gives bigrams, and `n = 3` gives trigrams.

Usage

```
ngram_frequency(text, n = 2L, stop_words = NULL, top_n = 30L)
```

Arguments

`text`	Character vector of responses (raw or already cleaned by `clean_text_responses()`).
`n`	Integer. N-gram size. Default `2` (bigrams).
`stop_words`	Character vector of words to exclude, or `NULL` to use the built-in English list, or `character(0)` for no filtering.
`top_n`	Integer. Maximum number of n-grams to return, most frequent first. Default `30`.

Details

An n-gram never straddles a removed stop word. Where "but" and "not" are filtered, "clean but not comfortable" yields no bigram at all, because "clean" and "comfortable" were never next to each other. This is what makes the output phrases respondents actually wrote.

Value

A data.frame with columns `term` (the space-joined n-gram), `n`, and `pct`.

Examples

```
demo <- sframe_demo_data()
cleaned <- clean_text_responses(demo$responses, "comments")
head(ngram_frequency(cleaned, n = 2, top_n = 10))
```

`outlier_report` *Flag univariate and multivariate outliers*

Description

Uses transparent screening rules for numeric survey response variables. The report supports data review before modelling, not automatic deletion.

Usage

```
outlier_report(
  data,
  variables = NULL,
  method = c("zscore", "iqr", "mahalanobis"),
  z_cut = 3,
  iqr_multiplier = 1.5,
  p_cut = 0.975
)
```

Arguments

`data`	A data.frame.
`variables`	Character vector of numeric variables to screen. When `NULL`, all numeric columns are used.
`method`	Outlier rule. `"zscore"` flags absolute z scores above `z_cut`, `"iqr"` flags values outside Tukey fences, and `"mahalanobis"` flags rows above the chi-square cutoff for the selected variables.
`z_cut`	Numeric cutoff for `"zscore"`. Defaults to `3`.
`iqr_multiplier`	Numeric multiplier for `"iqr"` fences. Defaults to `1.5`.
`p_cut`	Probability cutoff for `"mahalanobis"`. Defaults to `0.975`.

Value

An object of class `sframe_outlier_report` with the method, screened variables, a result table, flagged row numbers, and a reporting prompt.

Examples

```
demo <- sframe_demo_data()
outliers <- outlier_report(
  demo$responses,
  variables = c("dm_1", "dm_2", "sat_1"),
  method = "zscore"
)
outliers$flagged_rows
```

plot.sframe_analysis_results *Plot analysis-plan results*

Description

Draws the charts that `run_analysis_plan()` attaches when called with `plots = TRUE`. With `which` supplied, returns that single chart. With `which` omitted, prints every attached chart in queue order and returns the list invisibly. Regression diagnostic panels stay on the result's `diagnostic_plots` element and are not drawn here.

Usage

```
## S3 method for class 'sframe_analysis_results'
plot(x, ..., which = NULL)
```

Arguments

`x`	An `sframe_analysis_results` object from `run_analysis_plan()`.
`...`	Ignored.
`which`	A research-question number or a plan block id selecting one chart, or NULL for all.

Value

A ggplot2 object when `which` is supplied, otherwise an invisible named list of ggplot2 objects keyed by plan block id.

posthoc_report *Post-hoc and pairwise comparison report*

Description

Post-hoc and pairwise comparison report

Usage

```
posthoc_report(
  data,
  method = c("anova", "kruskal_wallis", "chi_square", "cochran_q"),
  outcome = NULL,
  group = NULL,
  table_vars = NULL,
  measures = NULL,
  correction = c("holm", "bonferroni", "BH")
)
```

Arguments

data	A data.frame.
method	Comparison family. Supports "anova", "kruskal_wallis", "chi_square", and "cochran_q".
outcome	Outcome variable for group comparisons.
group	Grouping variable for group comparisons.
table_vars	Two categorical variables for chi-square residuals and pairwise proportion tests.
measures	Repeated binary measures for pairwise McNemar tests.
correction	Multiple-comparison correction.

Value

An object of class sframe_posthoc_report.

Examples

```
demo <- sframe_demo_data()
pr <- posthoc_report(demo$responses, method = "kruskal_wallis",
                     outcome = "sat_1", group = "visit_type")
pr$tables$pairwise_wilcox
```

---

preview_demo *Preview a demo survey in one line*

Description

Loads a bundled demo with the standard branding and opens it as a respondent would see it, either as a standalone page in the browser or on SurveyStudio's preview screen. The preview has its collector removed, so trying it is safe.

Usage

```
preview_demo(
  name = "two_group",
  where = c("browser", "studio"),
  open = interactive()
)
```

Arguments

name	Character. The demo to preview. sframe_demos() lists them. Defaults to "two_group".
where	One of "browser", a standalone HTML page written by export_static_survey(), or "studio", the preview screen of launch_studio().
open	Logical. Whether to open the browser. Defaults to interactive().

Value

For "browser", the path of the HTML page, invisibly. For "studio", what launch_studio() returns.

See Also

sframe_demo(), export_static_survey(), launch_studio()

Examples

```
path <- preview_demo("two_group", open = FALSE)
file.exists(path)
```

---

print.sf_branch *Print an sf_branch object*

Description

Print an sf_branch object

Usage

```
## S3 method for class 'sf_branch'
print(x, ...)
```

Arguments

x	An object of class sf_branch.
...	Ignored. Present for S3 consistency.

Value

x, invisibly.

Examples

```
br <- sf_branch("q2", depends_on = "q1", operator = "==",
                value = "yes", action = "show")
print(br)
```

print.sf_check *Print an sf_check object*

Description

Print an sf_check object

Usage

```
## S3 method for class 'sf_check'
print(x, ...)
```

Arguments

x	An object of class sf_check.
...	Ignored. Present for S3 consistency.

Value

x, invisibly.

Examples

```
ck <- sf_check("attn1", item_id = "q5", type = "attention",
               pass_values = 3)
print(ck)
```

print.sf_choices *Print an sf_choices object*

Description

Print an sf_choices object

Usage

```
## S3 method for class 'sf_choices'
print(x, ...)
```

Arguments

x	An object of class sf_choices.
...	Ignored. Present for S3 consistency.

Value

x, invisibly.

Examples

```
cs <- sf_choices("agree5", 1:5,
                 c("Strongly disagree", "Disagree", "Neutral",
                   "Agree", "Strongly agree"))
print(cs)
```

---

print.sf_item *Print an sf_item object*

Description

Print an sf_item object

Usage

```
## S3 method for class 'sf_item'
print(x, ...)
```

Arguments

x	An object of class sf_item.
...	Ignored. Present for S3 consistency.

Value

x, invisibly.

Examples

```
it <- sf_item("q1", "How satisfied are you?", type = "likert",
              choice_set = "agree5")
print(it)
```

print.sf_model *Print an sf_model object*

Description

Print an sf_model object

Usage

```
## S3 method for class 'sf_model'
print(x, ...)
```

Arguments

x	An object of class sf_model.
...	Ignored. Present for S3 consistency.

Value

x, invisibly.

print.sf_scale *Print an sf_scale object*

Description

Print an sf_scale object

Usage

```
## S3 method for class 'sf_scale'
print(x, ...)
```

Arguments

x	An object of class sf_scale.
...	Ignored. Present for S3 consistency.

Value

x, invisibly.

Examples

```
sc <- sf_scale("sat", "Satisfaction", items = c("q1", "q2", "q3"))
print(sc)
```

print.sframe *Print an sframe instrument object*

Description

Displays a compact summary of an `sframe` instrument object, showing the title, version, item count, scale count, and validation status.

Usage

```
## S3 method for class 'sframe'
print(x, ...)
```

Arguments

`x`	An object of class `sframe`.
`...`	Ignored. Present for S3 consistency.

Value

`x`, invisibly.

Examples

```
item <- sf_item("q1", "How satisfied are you?", type = "likert",
                choice_set = "agree5")
instr <- sf_instrument("My Survey", components = list(item))
print(instr)
```

quality_report *Generate a data quality report for survey responses*

Description

Evaluates collected response data against the instrument specification and produces a structured quality report. The report covers attention check performance, completion time, straight-lining within scale blocks, item-level missingness, respondent-level missingness, and duplicate respondent IDs where supplied.

Usage

```
quality_report(
  data,
  instrument,
  respondent_id = NULL,
  submitted_at = NULL,
  started_at = NULL,
  time_min = NULL,
  straightline_scales = TRUE,
  straightline_min_items = 4L,
  missing_threshold = 0.2
)
```

Arguments

data	A tibble or data.frame of responses, typically produced by read_responses().
instrument	An sframe object created by sf_instrument().
respondent_id	Character or NULL. The column name holding unique respondent identifiers. Used for duplicate detection.
submitted_at	Character or NULL. The column name holding submission timestamps. Used for completion time analysis.
started_at	Character or NULL. The column name holding survey start timestamps. When NULL, quality_report() looks for a recognised start-time column automatically.
time_min	Numeric or NULL. Minimum acceptable completion time in seconds. Respondents with a submission time below this threshold are flagged as speeders when timing data are available.
straightline_scales	Logical. Whether to check for straight-lining within each defined scale block. Defaults to TRUE.
straightline_min_items	Integer. The minimum number of items a scale must have before it is checked for straight-lining. Defaults to 4. A respondent who gives the identical response to every item in a 2-item scale has done exactly what a genuinely consistent respondent does, this is not evidence of inattention on its own, and checking scales that short flags a large share of honest respondents (see the worked example in vignette("surveyframe"), where 3 two-item scales alone drove a 91 percent flag rate before this threshold existed). Set to 2 to restore the previous, more permissive behaviour. Many published scales have 3 items, and the default leaves them unchecked. Set 3 to check them, and read a flag on a 3-item scale as a prompt to look at the respondent, since identical answers to 3 items are common among attentive respondents. The printed report names each scale it skipped and the setting that would check it. In an analysis plan, pass it on the quality block as options = list(straightline_min_items = 3).

`missing_threshold`
: Numeric. The proportion of missing item responses above which a respondent is flagged. Defaults to `0.2`.

Details

Timing analysis is available when the data contain a submission timestamp column and either an explicit `started_at` column or one of the recognised defaults: `started_at`, `start_time`, `started`, or `.started_at`.

Value

An object of class `sframe_quality_report`, a named list with elements: `summary`, `attention`, `timing`, `straightline`, `missing`, and `duplicates`. Use `print()` for a formatted summary.

See Also

`sf_check()`, `read_responses()`, `score_scales()`

Examples

```
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
responses <- read_responses(
  system.file("extdata", "tourism_services_responses.csv",
              package = "surveyframe"),
  instr,
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  meta_cols = "started_at"
)
qr <- quality_report(
  responses,
  instr,
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  started_at = "started_at",
  straightline_scales = FALSE
)
print(qr)
```

---

read_bibtex *Read references from BibTeX*

Description

Parses BibTeX, the format Google Scholar, Zotero, Mendeley and most journal pages export, into `sf_reference()` records. Covers the common entry types (`@article`, `@book`, `@incollection`, `@inbook`, `@inproceedings`, `@techreport`, `@phdthesis`, `@mastersthesis`, `@misc`, `@online`, `@software`, `@dataset`, `@unpublished`) and the common fields. Accented letters written as LaTeX commands are converted. Base R alone.

Usage

```
read_bibtex(file = NULL, text = NULL, cited_by = NULL)
```

Arguments

`file`	Path to a `.bib` file.
`text`	BibTeX as a character string, used instead of `file`, for example an entry pasted from Google Scholar.
`cited_by`	Optional `cited_by` targets given to every entry read, see `sf_reference()`.

Value

A list of `sf_reference` objects, named by citation key. An entry that fails to parse is skipped with a warning naming it. A key used by more than one entry is an error that names every repeated key. Entries may be written `@article{key, ...}` or `@article(key, ...)`.

Examples

```
refs <- read_bibtex(text = "@article{cronbach_1951,
  author  = {Cronbach, Lee J.},
  title   = {Coefficient alpha and the internal structure of tests},
  journal = {Psychometrika},
  year    = {1951},
  volume  = {16},
  number  = {3},
  pages   = {297--334}
}")
names(refs)
```

---

read_responses *Read and validate survey responses*

Description

Loads survey response data and checks that it conforms to the instrument specification. Column names in the response file must match item IDs defined in the instrument. Non-item columns are allowed only when declared through `respondent_id`, `submitted_at`, or `meta_cols`.

Usage

```
read_responses(
  x,
  instrument,
  respondent_id = NULL,
  submitted_at = NULL,
  meta_cols = NULL,
  strict = TRUE,
  delim = NULL
)
```

Arguments

x	A file path to a delimited text file, a data.frame, or a tibble.
instrument	An sframe object created by sf_instrument().
respondent_id	Character or NULL. The name of the column containing unique respondent identifiers. If NULL, no respondent ID column is expected.
submitted_at	Character or NULL. The name of the column containing submission timestamps. The metadata columns surveyframe's own collectors write, respondent_id, response_id, started_at and submitted_at, are recognised without being declared: a file this package collected reads back without naming the columns it wrote. Anything else outside the instrument still has to be declared, or strict = FALSE used.
meta_cols	Character vector or NULL. Additional column names, outside the item IDs, to retain (for example, condition assignment or source URL).
strict	Logical. When TRUE (default), a column outside the declared item IDs, their expansion columns and the metadata columns is an error, naming the columns. When FALSE, such columns are kept, placed last, with a warning.
delim	NULL (default), ",", ";" or "\t". The field separator of a file. When NULL it is detected from the header line as a comma, semicolon or tab, and a message names the separator when it is not a comma. A semicolon file is read the way Excel writes one in a comma decimal locale, so a comma is the decimal mark in its numeric columns (3,5 reads as 3.5). A byte order mark is ignored. Not used for data frames.

Value

A data.frame with columns ordered as: metadata columns first, then item columns in instrument order, each item followed by its expansion columns, then any undeclared columns kept under strict = FALSE. To keep an extra column under strict = TRUE, name it in meta_cols, or select the columns you need before reading.

Columns

A single-answer item has one column named by its ID. A matrix, ranking, multiple-choice or decision item has one column per row, option, pair or criterion, named item__sub, and each is checked: a battery with some of its columns absent is reported by name. Two columns with the same name are refused, since one would otherwise be lost.

Values

A CSV file is read as text, so identifiers such as `001`, dates and text answers arrive exactly as written, a literal `NA` included. Columns of items with numeric responses (numeric, slider and rating items, choice items whose codes are all numbers, and ranking, multiple-choice and decision expansion columns) are then converted to numbers, with an empty cell or `NA` read as missing. A column that does not convert cleanly is kept as text. Data frames go through the same conversion, so a CSV file and a data frame holding the same responses read the same.

See Also

`quality_report()`, `score_scales()`

Examples

```
responses <- read_responses(
  x = system.file("extdata", "tourism_services_responses.csv",
                  package = "surveyframe"),
  instrument = read_sframe(
    system.file("extdata", "tourism_services_demo.sframe",
                package = "surveyframe")
  ),
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  meta_cols = "started_at"
)
head(responses[, c("respondent_id", "visit_type", "dm_1")])
```

---

read_sframe *Read an instrument from a .sframe file*

Description

Reads a `.sframe` JSON file, or the same JSON held as text, and reconstructs an `sframe` instrument object. The SHA-256 integrity hash is verified by default, and content that does not match its hash is refused. The hash covers a canonical form of the content, so it detects a change to the content of a written file, and it ignores whitespace and key order. It is unsigned, so anyone who edits a file can also recompute it.

Usage

```
read_sframe(path = NULL, validate = TRUE, text = NULL, verify = TRUE)
```

Arguments

`path`	Character. The path to a `.sframe` file. Leave it `NULL` and give `text` to read a payload held in memory.

validate — Logical. Whether to validate the loaded instrument with validate_sframe(). Defaults to TRUE. This controls structural validation only. verify controls the integrity hash.

text — Character. The .sframe JSON itself, for example a payload a host keeps in a database, read with no temporary file. One string, or lines to be joined.

verify — Logical. Whether to refuse content whose hash is missing or does not match. Defaults to TRUE. With FALSE, such content is read with a warning of class sframe_unverified_warning and the instrument is marked unverified (see the Unverified reads section).

Details

The instrument remembers the content and amendment log it was read with, so write_sframe() can refuse an undisclosed revision.

Value

An sframe object.

Unverified reads

verify = FALSE is for a host holding a payload it cannot vouch for, which it would otherwise rebuild by hand and lose fields doing so. The result carries the attribute sframe_verified, FALSE (it is TRUE on every verified read), and print() says so. write_sframe() refuses to write an unverified instrument unless new_instrument = TRUE declares it new, since nothing ties its content to a state anyone wrote. The exception is one written back unchanged with the amendment log it was read with. No amendment can start from it: amend_sframe() refuses it, and the write refuses any entry appended to its log.

See Also

write_sframe(), validate_sframe()

Examples

```
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
print(instr)
```

`read_sheet_responses` *Read survey responses from a Google Sheet*

Description

Reads response data collected by the surveyframe Google Apps Script endpoint and returns a validated data frame ready for the surveyframe analysis pipeline.

Usage

```
read_sheet_responses(
  sheet_id,
  instrument,
  sheet_name = "Responses",
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  meta_cols = NULL
)
```

Arguments

`sheet_id`	Character. The Google Sheet ID or full URL.
`instrument`	An `sframe` object.
`sheet_name`	Character. The name of the sheet tab holding responses. Defaults to `"Responses"`.
`respondent_id`	Character or NULL. Column holding respondent IDs. Defaults to `"respondent_id"`.
`submitted_at`	Character or NULL. Column holding submission timestamps. Defaults to `"submitted_at"`.
`meta_cols`	Character vector or NULL. Additional sheet columns to accept as metadata without a warning, for example bridge fields a host application appends to each submission. `"started_at"` is always included.

Value

A `data.frame` validated against the instrument, ready for `quality_report()`, `score_scales()`, and `reliability_report()`.

See Also

`export_google_sheet()`, `read_responses()`, `quality_report()`

Examples

```
## Not run:
responses <- read_sheet_responses(
  sheet_id   = "your-sheet-id",
  instrument = instr
)
qr <- quality_report(responses, instr, respondent_id = "respondent_id")

## End(Not run)
```

`reliability_report` *Estimate scale reliability from item responses*

Description

Produces Cronbach's alpha and McDonald's omega for each scale defined in the instrument, along with the number of items and sample size.

Usage

```
reliability_report(data, instrument, scales = NULL, alpha = TRUE, omega = TRUE)
```

Arguments

`data`	A `tibble` or `data.frame` of responses. Item columns must be present.
`instrument`	An `sframe` object.
`scales`	Character vector or NULL. A subset of scale IDs to analyse. When NULL (default), all scales in the instrument are included.
`alpha`	Logical. Whether to compute Cronbach's alpha. Defaults to `TRUE`.
`omega`	Logical. Whether to compute McDonald's omega. Defaults to `TRUE`.

Value

An object of class `sframe_reliability_report`, a list with one element per scale. Each element is a list of statistics and a summary tibble.

See Also

`sf_scale()`, `item_report()`

Examples

```
if (requireNamespace("psych", quietly = TRUE)) {
  demo <- sframe_demo_data()
  rr <- reliability_report(demo$responses, demo$instrument, omega = FALSE)
  print(rr)
}
```

render_report *Render a reproducible survey report*

Description

Generates an HTML report that includes the instrument codebook, data quality summary, reliability diagnostics, and analysis-plan content. When Quarto and the bundled template are available, the report is rendered through Quarto. Otherwise, surveyframe writes an internal HTML fallback so the reporting workflow still runs on machines without Quarto.

Usage

```
render_report(
  instrument,
  data = NULL,
  output_file = NULL,
  output_path = NULL,
  format = c("html", "pdf"),
  include_quality = TRUE,
  include_reliability = TRUE,
  include_codebook = TRUE,
  include_missing = TRUE,
  include_descriptives = TRUE,
  include_analysis = TRUE,
  include_models = TRUE,
  plot_palette = c("web", "print"),
  interpretations = NULL,
  show_code = TRUE,
  references = NULL,
  csl = NULL,
  title = NULL,
  authors = NULL,
  figures = getOption("surveyframe.figures"),
  figure_dir = NULL,
  open = interactive()
)
```

Arguments

instrument	An sframe object.
data	A tibble or data.frame of responses, or NULL to generate a codebook-only report.
output_file	Character or NULL. The output file path. When NULL, a temporary file is written and its path returned.
output_path	Character or NULL. Alias for output_file. If both are supplied, output_file takes precedence.

format Character. Output format: "html" (default) or "pdf". PDF output renders the HTML report and prints it through pagedown::chrome_print(), which requires the pagedown package (in Suggests) and a local Chrome or Chromium installation. The HTML path is unchanged.

include_quality
Logical. Whether to include the data quality report. Requires data. Defaults to TRUE.

include_reliability
Logical. Whether to include reliability diagnostics. Requires data. Defaults to TRUE.

include_codebook
Logical. Whether to include the instrument codebook. Defaults to TRUE.

include_missing
Logical. Whether to include the missing-data report. Requires data. Defaults to TRUE.

include_descriptives
Logical. Whether to include descriptive statistics. Requires data. Defaults to TRUE.

include_analysis
Logical. Whether to include analysis-plan results when data are supplied and the instrument has an analysis_plan.

include_models Logical. Whether to include saved model JSON and generated syntax blocks. Defaults to TRUE.

plot_palette One of "web" (brand colours, for on-screen reading) or "print" (black, grey, and white, for a journal-ready or print-friendly report). Applied to every chart the report embeds. See sframe_brand().

interpretations
Named list or NULL. Written interpretations keyed by analysis-plan block id, added after the results are known. Each block's section prints the hypotheses and planned decision rule it declared before its result, with or without an entry. An entry prints after the result under "Interpretation (written after the results)", in place of any interpretation the block holds. Interpretations are report content only and are never written into the instrument.

show_code Logical. Whether each result carries a folded "Show R code" block with the statistical call that produced it, built by analysis_syntax() from the same resolved specification the analysis ran. Defaults to TRUE. A method analysis_syntax() does not cover shows no block, rather than a guess at the call.

references Optional further sf_reference() records, for example the works a host cites in the written interpretations, which are cited in text as [@id]. The instrument's own references (add_references()) and the methods' references are always included.

csl Optional path to a Citation Style Language file for the References section. Defaults to APA 7th edition, bundled with the package. Any style from the Zotero style repository works.

title Character or NULL. The report's title. Defaults to the instrument's title.

authors	Character vector or NULL. The report's authors, shown under the title. Defaults to the instrument's authors, and is left out when there are none.
figures	List or NULL. Size, resolution and format for every figure in the report: any of width and height in inches, dpi, and format, "png" or "svg" inside the report. For example list(width = 6.5, height = 4, dpi = 300). Absent entries keep each engine's own default. Defaults to the surveyframe.figures option, so a host can set it once with options(surveyframe.figures = list(dpi = 300)).
figure_dir	Character or NULL. A directory to write each analysis plan chart into as its own file, through sframe_save_figures(), at the size, resolution and format in figures, where format may also be "tiff", "jpeg" or "pdf". Needs data. Defaults to 300 dpi at 7 by 4.5 inches where figures names none.
open	Logical. Whether to show the finished report: in the viewer pane of RStudio, Positron or VS Code, and in the web browser otherwise. Defaults to TRUE in an interactive session and FALSE in a script. Nothing opens during a knitr or Quarto render.

Value

The output file path, invisibly.

References

The Quarto report cites in text, author and year, and lists every work it cites once, in a References section at the end, formatted by pandoc's citation processor in the chosen style. That covers the methods' own references, the sources the instrument cites for its items, scales, plan blocks and models, and references. The built-in HTML engine, used when Quarto is unavailable, lists the methods' references the same way.

See Also

codebook_report(), quality_report(), reliability_report()

Examples

```
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
responses <- read_responses(
  system.file("extdata", "tourism_services_responses.csv",
              package = "surveyframe"),
  instr,
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  meta_cols = "started_at"
)

old <- options(surveyframe.use_quarto = FALSE)
out <- tryCatch(
  render_report(
```

```
    instr,
    data = responses,
    output_file = tempfile(fileext = ".html"),
    include_reliability = FALSE,
    include_analysis = FALSE
  ),
  finally = options(old)
)
file.exists(out)
```

`render_result` *Render one analysis result as the report does*

Description

Takes one result from `run_analysis_plan()` and returns what the report shows for it: the formatted table with its caption and note, any second table `sframe_result_supplement()` gives, the APA line, and the chart the report picks for that method, with the instrument's labels. The table, the chart and their choice per method come from the code `render_report()` uses, so a host rendering results one at a time and the report agree.

Usage

```
render_result(
  result,
  data,
  instrument = NULL,
  palette = c("web", "print"),
  format = c("html", "png", "svg"),
  width = 7,
  height = 4.5,
  dpi = 150
)
```

Arguments

`result`	One element of the list `run_analysis_plan()` returns.
`data`	The response data the result was computed from. The chart is drawn from it, with the scales scored first as `run_analysis_plan()` scores them when `instrument` is given.
`instrument`	The `sframe` the plan belongs to, or `NULL`. Gives the chart the item and choice labels the tables use.
`palette`	`"web"` (default) or `"print"`, the report's 2 palettes. A palette given explicitly rebuilds the chart with it. Left out, the chart attached to the result is used as it is.
`format`	`"html"` (default) returns an HTML fragment with the chart embedded. `"png"` and `"svg"` write the chart to a file as well, and return its path in `chart_file`.

`width`, `height`, `dpi`
: The chart's size in inches and its resolution, for the embedded or written chart. Defaults 7 by 4.5 at 150 dpi.

Value

A list with `block_id`, `test`, `caption`, `apa`, `table` (a data frame), `note`, `supplement` (a list with `table` and `caption`, or `NULL`), `chart` (a `ggplot`, or `NULL` where the method has none or ggplot2 is absent), `html` (the fragment, for `format = "html"`), `chart_file` (for `"png"` and `"svg"`) and `syntax` where the result has one. A result that failed carries its `error`, and `html` shows it.

See Also

`run_analysis_plan()`, `render_report()`, `sframe_save_figures()`

Examples

```
d <- sframe_demo("two_group")
res <- run_analysis_plan(d$responses, d$instrument)
out <- render_result(res[[1]], d$responses, d$instrument)
out$caption
out$table
```

---

render_results *Render analysis results to a formatted HTML report*

Description

Generates a self-contained HTML report from the output of `run_analysis_plan()`. Each section corresponds to one research question and includes the APA-formatted statistical result, an interpretation space, and a reference list.

Usage

```
render_results(
  results = NULL,
  instrument,
  output_file = NULL,
  output_path = NULL,
  citation_format = c("apa", "ama", "vancouver"),
  title = NULL,
  interpretations = NULL
)
```

Arguments

`results`	An `sframe_analysis_results` object from `run_analysis_plan()`.
`instrument`	An `sframe` object.
`output_file`	Character or NULL. Path to the output HTML file. When NULL, a temporary file is written and its path returned.
`output_path`	Character or NULL. Alias for `output_file`.
`citation_format`	Character. Reference format. One of `"apa"`, `"ama"`, or `"vancouver"`. Defaults to `"apa"`.
`title`	Character or NULL. Report title. Defaults to the instrument title with " – Results" appended.
`interpretations`	Named list or NULL. Written interpretations keyed by analysis-plan block id, added after the results are known. A block with an entry shows that text in its Interpretation section, labelled as written after the results. The hypotheses and planned decision rule a block declared print before its result, with or without an entry. Interpretations are report content only and are never written into the instrument.

Value

The output file path, invisibly.

See Also

`run_analysis_plan()`, `render_report()`

Examples

```
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
responses <- read_responses(
  system.file("extdata", "tourism_services_responses.csv",
              package = "surveyframe"),
  instr,
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  meta_cols = "started_at"
)

results <- run_analysis_plan(responses, instr)
out <- render_results(results, instr,
                      output_file = tempfile(fileext = ".html"))
file.exists(out)
```

render_survey *Render a survey from an instrument object*

Description

Launches a Shiny survey with a welcome page, configurable header, all item types, branching logic, required-field enforcement, progress tracking, standard and conversational (one-question-at-a-time) display modes, and a customisable thank-you page. Responses can be persisted to CSV or passed to a callback.

Usage

```
render_survey(
  instrument,
  mode = c("shiny"),
  title = NULL,
  theme = NULL,
  save_responses = c("none", "csv"),
  output_path = NULL,
  on_submit = NULL
)
```

Arguments

instrument	An sframe object.
mode	Character. Deployment mode. Currently "shiny".
title	Character or NULL. Override for the survey title.
theme	Character or NULL. Hex colour for the survey theme.
save_responses	Character. "none" (default) or "csv".
output_path	Character or NULL. CSV path when save_responses = "csv".
on_submit	Function or NULL. Callback receiving the submitted row.

Value

A shiny.appobj.

See Also

launch_studio(), read_responses()

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
item  <- sf_item("sat_1", "How satisfied are you?",
                 type = "likert", choice_set = "ag5")
instr <- sf_instrument("My Survey", components = list(cs, item))
app <- render_survey(instr)
app <- render_survey(instr, save_responses = "csv",
                     output_path = tempfile(fileext = ".csv"))
```

`run_analysis_plan` *Run a pre-planned analysis from an instrument's analysis plan*

Description

Executes every analysis block defined in the instrument's `analysis_plan` slot against the supplied response data. Each block corresponds to one research question defined during instrument design in the SurveyBuilder. Results include APA-formatted statistics, effect sizes, interpretation prompts, and reporting references.

Usage

```
run_analysis_plan(
  data,
  instrument,
  scored = TRUE,
  plots = FALSE,
  plot_palette = c("web", "print"),
  seed = 20260828L,
  strict = FALSE
)
```

Arguments

`data`	A `tibble` or `data.frame` of responses, typically produced by `read_responses()` or `read_sheet_responses()`.
`instrument`	An `sframe` object containing an `analysis_plan`.
`scored`	Logical. Whether to automatically score scales before running the analysis. Defaults to `TRUE`.
`plots`	Logical. When `TRUE` and ggplot2 is installed, supported blocks gain a `$plot` element holding a brand-styled ggplot object: bar charts for frequency and chi-square blocks, scatter plots with a regression overlay for correlation and linear-regression blocks. Defaults to `FALSE`.

plot_palette	One of "web" (brand colours, for on-screen use) or "print" (black, grey, and white, for journal-ready print figures). Applied to every plot attached when plots = TRUE. See sframe_brand().
seed	Integer or NULL. The random seed the analysis runs under. Defaults to a fixed value so the same instrument and the same data give the same answer every time, which is what makes a rendered report usable as an audit artefact. Every bootstrap confidence interval in the plan, and the parallel analysis behind the EFA family, depend on it. Pass NULL for the unseeded behaviour of releases before 0.4.1. The caller's own random stream is restored afterwards, so seeding here does not affect anything that runs later.
strict	Logical. When TRUE, an error is raised if any block fails or scale scoring fails, listing each failure. When FALSE, the default, failures are kept in the results and counted in their status.

Value

An object of class sframe_analysis_results, a list with one element per analysis block, named by block id. Its status attribute holds blocks, succeeded, failed, failed_blocks and scoring, the outcome of scoring scales before analysis. A normal return can hold failed blocks, so check attr(results, "status")$failed or use strict = TRUE. Each block element Each element contains the test result, APA string, interpretation prompt, and reporting-reference metadata. Inferential blocks also carry a $table data frame suitable for knitr::kable(). Pass to render_results() to generate a formatted report.

See Also

render_results(), read_sheet_responses()

Examples

```
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
responses <- read_responses(
  system.file("extdata", "tourism_services_responses.csv",
              package = "surveyframe"),
  instr,
  respondent_id = "respondent_id",
  submitted_at = "submitted_at",
  meta_cols = "started_at"
)

results <- run_analysis_plan(responses, instr)
print(results)
```

`sample_size_plan` *Sample-size and power planning helper*

Description

Estimates a total sample size for a planned analysis. The targets use 3 different methods, and the result says which one applied.

Usage

```
sample_size_plan(
  type = c("proportion", "mean", "correlation", "t_test", "anova",
    "regression", "sem"),
  margin_error = NULL,
  sd = NULL,
  p = 0.5,
  r = NULL,
  alpha = 0.05,
  power = 0.8,
  groups = 2L,
  predictors = NULL,
  d = NULL,
  f = NULL,
  f2 = NULL
)
```

Arguments

`type`	Planning target: `"proportion"`, `"mean"`, `"correlation"`, `"t_test"`, `"anova"`, `"regression"`, or `"sem"`.
`margin_error`	Margin of error for mean/proportion planning.
`sd`	Standard deviation for mean planning.
`p`	Expected proportion.
`r`	Expected correlation. Defaults to 0.30 with a warning.
`alpha`	Significance level.
`power`	Desired power, for the power calculations.
`groups`	Number of groups for ANOVA planning. A t test has 2.
`predictors`	Number of predictors for regression planning.
`d`	Expected Cohen's d for a t test. Defaults to 0.5 with a warning.
`f`	Expected Cohen's f for ANOVA. Defaults to 0.25 with a warning.
`f2`	Expected Cohen's f squared for regression. When `NULL`, a rule of thumb is returned in place of a power calculation.

Value

An sframe_sample_size_plan list holding type, estimated_n (total sample size), method ("power", "precision", "rule_of_thumb" or "none"), alpha, power, effect_size, warnings, advisory and prompt.

Power calculations

"t_test", "anova" and "correlation" are power calculations, so alpha, power and the expected effect size all change the result. The t test uses stats::power.t.test() for 2 independent groups with Cohen's d. ANOVA uses stats::power.anova.test() with Cohen's f. Correlation uses the Fisher z approximation with r. "regression" is a power calculation when f2 is supplied, from the noncentral F distribution for the overall test of predictors predictors.

When the effect size is left NULL, a conventional medium effect is assumed (d 0.5, f 0.25, r 0.30) and a warning names it. An assumed effect is a placeholder. Supply the effect you expect from prior studies or a pilot.

Precision targets and rules of thumb

"proportion" and "mean" size a confidence interval to a margin of error, and use alpha for its confidence level. power has no bearing on them. "regression" without f2 returns the larger of 2 published rules of thumb, 50 + 8k and 104 + k, which ignore alpha and power. "sem" returns no estimate. Both say so in the returned warnings.

Examples

```
plan <- sample_size_plan("t_test", d = 0.5, power = 0.80)
plan$estimated_n
plan$method
```

---

score_scales *Score defined scales from survey responses*

Description

Applies scale scoring rules from the instrument to response data. Handles reverse coding, optional weighted composite score computation, and minimum valid item thresholds. Returns a data frame with one scored column per scale.

Usage

```
score_scales(data, instrument, keep_items = TRUE, keep_meta = TRUE)
```

Arguments

data	A tibble or data.frame of responses.
instrument	An sframe object.
keep_items	Logical. Whether to retain individual item columns in the output. Defaults to TRUE.
keep_meta	Logical. Whether to retain non-item columns (metadata) in the output. Defaults to TRUE.

Value

A data.frame with scored scale columns appended. Scale columns are named using the scale id.

See Also

sf_scale(), reliability_report()

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
i1    <- sf_item("sat_1", "Item 1", type = "likert",
                 choice_set = "ag5", scale_id = "sat")
i2    <- sf_item("sat_2", "Item 2", type = "likert",
                 choice_set = "ag5", scale_id = "sat")
i3    <- sf_item("sat_3", "Item 3 (reverse)", type = "likert",
                 choice_set = "ag5", scale_id = "sat", reverse = TRUE)
scale <- sf_scale("sat", "Satisfaction",
                  items = c("sat_1", "sat_2", "sat_3"), min_valid = 2L)
instr <- sf_instrument("Demo", components = list(cs, i1, i2, i3, scale))

responses <- data.frame(
  sat_1 = c(4, 5, 3),
  sat_2 = c(4, 4, 3),
  sat_3 = c(2, 1, 3),
  stringsAsFactors = FALSE
)

scored <- score_scales(responses, instr)
scored$sat
```

---

sem_lavaan_syntax *Generate lavaan CB-SEM syntax*

Description

Generate lavaan CB-SEM syntax

Usage

```
sem_lavaan_syntax(model, instrument = NULL, standardised = TRUE)
```

Arguments

model	An sf_model() object of type "cb_sem".
instrument	Optional sframe object for indicator validation.
standardised	Logical. Adds a standardised-estimates fitting note.

Value

A lavaan syntax string.

Examples

```
m <- sf_model(
  "cb1", type = "cb_sem",
  constructs = list(
    sf_construct("sq", items = c("sq_1", "sq_2", "sq_3")),
    sf_construct("sat", items = c("sat_1", "sat_2")),
    sf_construct("bi", items = c("bi_1", "bi_2"))
  ),
  paths = list(sf_path("sq", "sat"), sf_path("sat", "bi")),
  indirect = list(sf_indirect("sq", through = "sat", to = "bi"))
)
cat(sem_lavaan_syntax(m))
```

---

seminr_syntax *Generate seminr PLS-SEM syntax*

Description

Generate seminr PLS-SEM syntax

Usage

```
seminr_syntax(model, data_name = "data", nboot = NULL, seed = 123)
```

Arguments

model	An sf_model() object of type "pls_sem".
data_name	Name of the data object in generated R code.
nboot	Number of bootstrap samples.
seed	Random seed for bootstrap syntax.

Value

An R syntax string for seminr.

Examples

```
m <- sf_model(
  "pls1", type = "pls_sem",
  constructs = list(
    sf_construct("sq", items = c("sq_1", "sq_2", "sq_3")),
    sf_construct("sat", items = c("sat_1", "sat_2"))
  ),
  paths = list(sf_path("sq", "sat"))
)
cat(seminr_syntax(m))
```

`sensitivity_analysis` *Test how far a decision ranking moves when the weights are perturbed*

Description

Perturbs each criterion weight up and down by `delta`, renormalises the weight vector to sum to 1, reruns the same ranking method, and compares the perturbed ranking against the base ranking. A ranking that survives this unchanged is one a reviewer can be told is robust to the weights. A ranking whose leader changes under a 5 percent nudge is not.

Usage

```
sensitivity_analysis(
  x,
  weights,
  criteria_types,
  method = "topsis",
  delta = 0.05,
  alternatives = NULL,
  criteria = NULL,
  ...
)
```

Arguments

`x`	Numeric performance matrix, alternatives in rows and criteria in columns.
`weights`	Numeric weight vector, one per criterion. Renormalised to sum to 1 before use. Unnamed weights are taken in the order of the criteria. Named weights are matched to the criteria by name, in any order, and a name that is not a criterion, or a criterion with no weight, is an error.
`criteria_types`	Character vector of `"benefit"` or `"cost"`, one per criterion.
`method`	Ranking method. One of `"topsis"`, `"vikor"`, `"moora"`, `"smart"`, `"waspas"`, `"promethee"`, or `"electre"`.
`delta`	Perturbation size as a proportion of the weight, default `0.05`. A weight of 0.40 with `delta = 0.05` is tested at 0.42 and 0.38 before renormalisation.

alternatives	Optional labels for the rows of x.
criteria	Optional labels for the columns of x.
...	Passed to the underlying method, for example v for VIKOR or lambda for WASPAS.

Value

An object of class sframe_sensitivity, a list with $table (one row per criterion and direction, carrying criterion, direction, rho, rank_changed, and top_changed), $base_ranks, $method, $delta, $stable, $degenerate, $n_perturbations, $n_effective and $n_failed. $stable is TRUE when at least one perturbation moved the weights and none of those that did changed the ranking. A perturbation that leaves the renormalised weights unchanged, as every one does for weights such as c(1, 0), is not effective and tests nothing. $degenerate marks a base ranking that places every alternative at the same rank, which no perturbation can move.

See Also

run_analysis_plan(), sframe_decision_options()

Examples

```
x <- matrix(c(4.1, 3.0, 210, 3.6, 4.5, 180, 4.8, 2.5, 260),
            nrow = 3, byrow = TRUE)
sa <- sensitivity_analysis(
  x,
  weights        = c(0.4, 0.3, 0.3),
  criteria_types = c("benefit", "benefit", "cost"),
  method         = "topsis",
  alternatives   = c("Alpha", "Basilica", "Coral"),
  criteria       = c("service", "location", "price")
)
sa$stable
as.data.frame(sa)
```

---

sf_accessors *Explore a surveyframe object*

Description

Accessors for the parts of an instrument, a codebook, or a report. They replace reaching into the object with $, which ties user code to the internal layout.

Details

What each one gives back depends on what it is asked. Given an instrument, the component accessors return the component objects as an sf_component_list, which prints as a list and is subset with [and [[. Given a codebook, the same verbs return the table the codebook already holds, a plain data frame with one row per item, scale, choice set, model or plan block.

Accessor	On an sframe	On an sframe_codebook
sf_meta()	list of metadata	list of metadata
sf_items()	sf_component_list of items	data frame of items
sf_scales()	sf_component_list of scales	data frame of scales
sf_choice_sets()	sf_component_list of choice sets	data frame of choice sets
sf_branches()	sf_component_list of branching rules	not available
sf_checks()	sf_component_list of checks	not available
sf_models()	sf_component_list of models	data frame of models
sf_plan()	list of plan blocks	data frame of plan blocks

as.data.frame() on an instrument gives its items as a table, which is one part of it, and a component list has no coercion of its own. For every table an instrument can produce, use codebook_report().

Value

A list for sf_meta() and sf_plan() on an instrument, an sf_component_list for the component accessors on an instrument, and a data frame for any of them on a codebook. See the table above.

See Also

as_sframe(), sf_problems(), sframe_validation

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
item  <- sf_item("sat_1", "The service met my expectations.",
                 type = "likert", choice_set = "ag5", scale_id = "sat")
scale <- sf_scale("sat", "Satisfaction", items = "sat_1")
instr <- sf_instrument("Demo Survey", components = list(cs, item, scale))

sf_meta(instr)$title
sf_items(instr)
sf_scales(instr)[["sat"]]
as.data.frame(instr)
```

---

sf_apa *Extract APA-formatted result summaries*

Description

Returns the report-ready sentence attached to each analysis result.

Usage

```
sf_apa(x, ...)
```

Arguments

x	An sframe_analysis_results, sframe_descriptives_report, sframe_missing_data_report, sframe_validity_report, or sframe_assumption_report object.
...	Passed to methods.

Value

A character vector containing one APA-formatted summary per result.

See Also

sf_report_accessors, run_analysis_plan()

Examples

```
results <- structure(
  list(RQ1 = list(apa = "Mean satisfaction was 4.20.")),
  class = c("sframe_analysis_results", "list")
)
sf_apa(results)
```

---

sf_branch *Define a branching rule*

Description

Creates a single-condition branching rule that shows or hides a survey item depending on the value of a preceding item. Only single-condition rules are supported. Multi-condition AND/OR logic is planned for a later release.

Usage

```
sf_branch(
  item_id,
  depends_on,
  operator = c("==", "!=", "%in%", ">", ">=", "<", "<="),
  value,
  action = c("show", "hide")
)
```

Arguments

item_id	Character. The id of the item whose visibility this rule controls.
depends_on	Character. The id of the item whose response value triggers this rule.
operator	Character. The comparison operator. One of "==", "!=", "%in%", ">", ">=", "<", or "<=".

value	The value to compare against the response to depends_on. For "%in%", supply a character or numeric vector.
action	Character. What to do when the condition is met. Either "show" (default) or "hide".

Value

An object of class sf_branch (a named list).

See Also

sf_instrument(), validate_sframe()

Examples

```
# Show an open-text follow-up only when the respondent selects "Other"
rule <- sf_branch(
  item_id    = "gender_other",
  depends_on = "gender",
  operator   = "==",
  value      = "other",
  action     = "show"
)
```

---

sf_branches *Get branching rules*

Description

Returns the rules that control whether conditional items are shown.

Usage

```
sf_branches(x, ...)
```

Arguments

x	A surveyframe object.
...	Passed to methods.

Value

An sf_component_list of branching rules.

See Also

sf_accessors

Examples

```
sf_branches(sframe_demo_data()$instrument)
```

`sf_check` *Define a design-time survey check*

Description

Specifies an attention, instructional, or trap check item at instrument design time. The check is stored in the instrument object and evaluated against collected response data by `quality_report()`. This function only defines the check. Evaluation happens later in `quality_report()`.

Usage

```
sf_check(
  id,
  item_id,
  type = c("attention", "instructional", "trap"),
  pass_values = NULL,
  fail_action = c("flag", "exclude"),
  label = NULL,
  notes = NULL
)
```

Arguments

`id`	Character. A unique identifier for this check.
`item_id`	Character. The `id` of the item used as the check. The item must be defined separately with `sf_item()` and included in the same instrument.
`type`	Character. The check type. One of: • `"attention"`: the item has a stated correct answer and flags respondents who answer incorrectly. • `"instructional"`: a manipulation check item used to test whether instructions were followed. • `"trap"`: an item designed to be selected only by inattentive respondents (e.g. "Please select Strongly agree for this item.").
`pass_values`	Vector or NULL. The response value or values that constitute a pass. For `"attention"` and `"instructional"` types, at least one value should be supplied. For `"trap"` types, this is the value that should NOT be selected.
`fail_action`	Character. What `quality_report()` does with respondents who fail this check. Either `"flag"` (mark in the report but retain) or `"exclude"` (mark for exclusion).
`label`	Character or NULL. An optional human-readable label for the check, used in the quality report.
`notes`	Character or NULL. Optional free-text notes about the purpose or rationale of this check.

Value

An object of class `sf_check` (a named list).

See Also

`sf_item()`, `sf_instrument()`, `quality_report()`

Examples

```
# An attention check: respondent must select 4
chk <- sf_check(
  id          = "attn_1",
  item_id     = "attention_check_q",
  type        = "attention",
  pass_values = 4,
  fail_action = "flag",
  label       = "Attention check 1"
)
```

---

sf_checks *Get response-quality checks*

Description

Returns declared attention and other response-quality checks.

Usage

```
sf_checks(x, ...)
```

Arguments

`x`	A surveyframe object.
`...`	Passed to methods.

Value

An sf_component_list of declared checks.

See Also

sf_accessors

Examples

```
sf_checks(sframe_demo_data()$instrument)
```

sf_choice_sets *Get choice sets*

Description

Returns the reusable value-and-label sets referenced by closed-response items.

Usage

```
sf_choice_sets(x, ...)
```

Arguments

x	A surveyframe object.
...	Passed to methods.

Value

An sf_component_list for an instrument or a choice-set data frame for a codebook.

See Also

sf_accessors

Examples

```
sf_choice_sets(sframe_demo_data()$instrument)
```

sf_choices *Define a reusable choice set*

Description

Creates a named set of response options that can be referenced by one or more items. Defining choices once and referencing them by `id` keeps the instrument consistent and reduces the risk of label mismatches across items that share the same response format.

Usage

```
sf_choices(
  id,
  values,
  labels,
  allow_other = FALSE,
  randomise = FALSE,
  ordered = FALSE
)
```

Arguments

id	Character. A unique identifier for this choice set. Referenced in the choice_set argument of sf_item().
values	Character or numeric vector. The stored values corresponding to each response option. Must have the same length as labels.
labels	Character vector. The display labels shown to respondents. Must have the same length as values.
allow_other	Logical. Whether to append an open-text "Other" option at the end of the choice list. Defaults to FALSE.
randomise	Logical. Whether to randomise the display order of options at render time. Defaults to FALSE.
ordered	Logical. Whether the options run in a meaningful order, from least to most or the reverse, as in a frequency or an education scale. SurveyBuilder and SurveyStudio then treat a single_choice item on the set as ordinal, which opens the methods for ordered categories. Defaults to FALSE, nominal. Written to the file only when TRUE, so every earlier set keeps the hash it had. A likert item is ordered by its type.

Value

An object of class sf_choices (a named list).

See Also

sf_item(), sf_instrument()

Examples

```
# A five-point agreement scale
agree5 <- sf_choices(
  id     = "agree5",
  values = 1:5,
  labels = c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree")
)

# An ordered set: a single_choice item on it is read as ordinal
often <- sf_choices(
  id      = "often",
  values  = 1:4,
  labels  = c("Rarely", "Sometimes", "Often", "Always"),
  ordered = TRUE
)
```

sf_component_list *A list of instrument components*

Description

The value returned by `sf_items()`, `sf_scales()`, `sf_choice_sets()`, `sf_branches()`, `sf_checks()` and `sf_models()`. It is a list of component objects named by their IDs, so a single component is reached with `[[`.

Value

`print()` returns `x` invisibly. `[` returns an `sf_component_list`.

Examples

```
item1 <- sf_item("q1", "First question", type = "text")
item2 <- sf_item("q2", "Second question", type = "text")
instr <- sf_instrument("Demo", components = list(item1, item2))

sf_items(instr)
sf_items(instr)[["q2"]]
```

sf_conjoint_design *Declare a choice-experiment (conjoint) design*

Description

Generates and stores a conjoint profile set and task schedule as part of the instrument's pre-declared contract. This is a design generator, not an estimator: it fixes what respondents will be shown, and it does not fit or analyse choice models.

Usage

```
sf_conjoint_design(
  id,
  attributes,
  method = c("full", "balanced", "random"),
  n_profiles = NULL,
  n_alternatives = 2L,
  n_tasks = NULL,
  blocks = 1L,
  seed = NULL,
  profiles = NULL,
  label = NULL
)
```

Arguments

id	Design identifier. Must start with a letter and contain only letters, numbers, and _ characters.
attributes	Named list of character vectors, one per attribute, giving that attribute's levels. At least 2 attributes, each with at least 2 levels.
method	One of "full", "balanced", or "random". Ignored when profiles is supplied.
n_profiles	Number of profiles to keep. Required for "balanced" and "random", ignored for "full".
n_alternatives	Alternatives shown per choice task, default 2.
n_tasks	Choice tasks per block. Defaults to as many whole tasks as the profile set supports.
blocks	Number of blocks the tasks are split across, default 1.
seed	Integer seed. Generated and stored when not supplied.
profiles	Optional data frame of pre-built profiles, one column per attribute. Supplying this bypasses generation and declares the design as given.
label	Human-readable label.

Details

The design is reproducible by construction. seed is always recorded, and generated when not supplied, so regenerating from the stored declaration returns the identical profiles and tasks.

Value

An object of class sf_conjoint_design with $profiles, $tasks (long format, one row per block, task, and alternative, ready for a choice model), $balance, and the declaration that produced them.

Choosing a method

"full" enumerates every combination, which is exact but grows fast: 4 attributes at 3 levels each is 81 profiles. "random" takes a seeded random subset of that size. "balanced" samples repeatedly and keeps the subset with the most even level spread and the weakest association between attributes.

"balanced" is a search, not a construction. It does not produce a catalogued orthogonal fractional factorial and does not claim the guarantees of one. The achieved balance is reported in $balance so the design can be inspected rather than trusted. A study needing a specific D-optimal or orthogonal design should generate it elsewhere and pass it in through profiles, which keeps the declaration in the contract either way.

See Also

sf_instrument()

Examples

```
design <- sf_conjoint_design(
  "hotel_dce",
  attributes = list(
    price    = c("50", "100", "150"),
    board    = c("room only", "breakfast"),
    distance = c("beachfront", "10 min walk")
  ),
  method = "balanced", n_profiles = 6, n_alternatives = 2, seed = 42
)
design$profiles
design$tasks
```

`sf_construct` *Define a latent or composite construct*

Description

Define a latent or composite construct

Usage

```
sf_construct(
  id,
  label = NULL,
  items = character(0),
  mode = c("reflective", "composite", "formative", "single_item"),
  weights = NULL
)
```

Arguments

`id`	Construct identifier. Must start with a letter and contain only letters, numbers, and _ characters.
`label`	Human-readable construct label.
`items`	Character vector of indicator item IDs.
`mode`	Measurement mode. One of `"reflective"`, `"composite"`, `"formative"`, or `"single_item"`.
`weights`	Optional indicator weights for later PLS-SEM planning.

Value

An object of class `sf_construct`.

Examples

```
sq <- sf_construct("sq", "Service Quality",
                   items = c("sq_1", "sq_2", "sq_3"))
sq$mode
sq$items
```

---

sf_covariance *Define a covariance between constructs*

Description

Define a covariance between constructs

Usage

```
sf_covariance(from, to, label = NULL)
```

Arguments

from	First construct ID.
to	Second construct ID.
label	Optional label.

Value

An object of class `sf_covariance`.

Examples

```
cov <- sf_covariance("sq", "sus", label = "cov1")
cov$from
cov$to
```

---

sf_flagged *Get rows flagged by response-quality checks*

Description

Pools failed attention checks, straight-lining, excess missingness, timing, and duplicate checks into one set of response-row positions.

Usage

```
sf_flagged(x, ...)
```

Arguments

x	An sframe_quality_report object.
...	Passed to methods.

Value

A sorted integer vector of unique response-row positions.

See Also

sf_report_accessors, quality_report()

Examples

```
demo <- sframe_demo_data()
qr <- quality_report(demo$responses, demo$instrument)
sf_flagged(qr)
```

---

sf_id *Get an instrument component ID*

Description

Reads the stable identifier used to refer to a component elsewhere in the instrument.

Usage

```
sf_id(x, ...)
```

Arguments

x	An sf_item(), sf_choices(), sf_scale(), sf_branch(), sf_check() or sf_model() object.
...	Passed to methods.

Value

A single character identifier.

See Also

sf_identity, sf_label()

Examples

```
sf_id(sf_item("q1", "Satisfaction", type = "numeric"))
```

sf_identity *The ID and label of an instrument component*

Description

The ID and label of an instrument component

Value

A single character string. `sf_label()` returns `""` when the component carries no label.

Examples

```
item <- sf_item("q1", "How satisfied are you?", type = "likert",
                choice_set = "agree5")
sf_id(item)
sf_label(item)
```

sf_indirect *Define an indirect effect path*

Description

Define an indirect effect path

Usage

```
sf_indirect(from, through, to, label = NULL)
```

Arguments

`from`	Source construct ID.
`through`	Character vector of mediator construct IDs.
`to`	Target construct ID.
`label`	Optional effect label.

Value

An object of class `sf_indirect`.

Examples

```
ind <- sf_indirect("sq", through = "sat", to = "bi", label = "mediation")
ind$through
```

`sf_instrument` *Create a survey instrument object*

Description

Assembles a survey instrument from its component objects. This is the top-level constructor for the `sframe` class. All other constructors (`sf_item()`, `sf_choices()`, `sf_scale()`, `sf_branch()`, `sf_check()`) produce components that are passed into this function via `components`.

Usage

```
sf_instrument(
  title,
  version = "0.1.0",
  description = NULL,
  authors = NULL,
  languages = "en",
  components = list(),
  render = NULL,
  analysis_plan = list(),
  models = list()
)
```

Arguments

`title`	Character. The title of the survey instrument.
`version`	Character. A semantic version string. Defaults to `"0.1.0"`.
`description`	Character or NULL. A brief description of the instrument and its intended population or purpose.
`authors`	Character vector or NULL. Author names, used in codebooks and reports.
`languages`	Character vector. Language codes for the instrument. Defaults to `"en"`. Multi-language support is planned for a later release.
`components`	List. A list of component objects created by the constructor family: `sf_item()`, `sf_choices()`, `sf_scale()`, `sf_branch()`, `sf_check()`, `sf_conjoint_design()`, and `sf_reference()`. Components are sorted by class automatically, so they can be supplied in any order.
`render`	List or NULL. Optional rendering hints passed to `render_survey()`, such as theme colour or progress bar visibility.
`analysis_plan`	List. Optional pre-planned analysis blocks created in the HTML SurveyBuilder Analyse mode.
`models`	List. Optional model specifications created with `sf_model()` or imported from a `.sframe` file.

Value

An object of class `sframe` with slots `meta`, `items`, `choices`, `scales`, `branching`, `checks`, `analysis_plan`, `models`, and `render`.

See Also

`sf_item()`, `sf_choices()`, `sf_scale()`, `sf_branch()`, `sf_check()`, `validate_sframe()`, `write_sframe()`

Examples

```
choices <- sf_choices("agree5", 1:5,
  c("Strongly disagree", "Disagree", "Neutral", "Agree", "Strongly agree"))

visitor_cs <- sf_choices("visitor", c("new", "returning"),
                         c("New visitor", "Returning visitor"))

item1 <- sf_item("sat_1", "The service met my expectations.",
                 type = "likert", choice_set = "agree5",
                 scale_id = "sat", required = TRUE)
item2 <- sf_item("sat_2", "I would recommend this service.",
                 type = "likert", choice_set = "agree5",
                 scale_id = "sat", required = TRUE)
item3 <- sf_item("visitor_type", "I am a",
                 type = "single_choice", choice_set = "visitor")

scale <- sf_scale("sat", "Satisfaction", items = c("sat_1", "sat_2"))

# The analysis_plan binds each research question to a statistical method
# and the variable roles it needs. Declare it before any data arrive.
plan <- list(
  list(
    id              = "RQ1",
    research_question = "Do new and returning visitors differ in satisfaction?",
    family          = "group_comparison",
    method          = "mann_whitney",
    roles           = list(group = "visitor_type", outcome = "sat"),
    options         = list(alpha = 0.05)
  )
)

instr <- sf_instrument(
  title         = "Service Quality Survey",
  version       = "1.0.0",
  components    = list(choices, visitor_cs, item1, item2, item3, scale),
  analysis_plan = plan
)
print(instr)
length(sf_plan(instr))
```

sf_is_valid *Test whether validation passed*

Description

Reads the overall pass/fail result without inspecting validation internals.

Usage

```
sf_is_valid(x, ...)
```

Arguments

x	An sframe_validation object.
...	Passed to methods.

Value

A single logical value.

See Also

sf_validation_accessors, `validate_sframe()`

Examples

```
v <- validate_sframe(sframe_demo_data()$instrument, strict = FALSE)
sf_is_valid(v)
```

sf_item *Define a survey item*

Description

Creates a single survey item object for inclusion in an `sframe` instrument. Items are the atomic units of a survey instrument. Every item must have a unique `id` within the instrument it is added to.

Usage

```
sf_item(
  id,
  label,
  type = c("likert", "single_choice", "multiple_choice", "numeric",
    "text", "textarea", "date", "matrix", "slider", "ranking", "rating",
    "pairwise_comparison", "criteria_weight", "section_break",
    "text_block"),
```

```
  required = FALSE,
  choice_set = NULL,
  scale_id = NULL,
  reverse = FALSE,
  help = NULL,
  placeholder = NULL,
  matrix_items = NULL,
  comparison_items = NULL,
  comparison_scale = NULL,
  slider_min = NULL,
  slider_max = NULL,
  slider_step = NULL,
  rating_max = NULL,
  rating_icon = NULL,
  date_min = NULL,
  date_max = NULL,
  section_intro = NULL,
  page = NULL
)
```

Arguments

id	Character. A unique identifier for this item. Used as the column name in response data. Must contain only letters, numbers, and _ characters.
label	Character. The question text or content displayed to the respondent.
type	Character. The response type. One of "likert", "single_choice", "multiple_choice", "numeric", "text", "textarea", "date", "matrix", "slider", "ranking", "rating", "pairwise_comparison", "criteria_weight", "section_break", or "text_block".
required	Logical. Whether the respondent must answer this item.
choice_set	Character or NULL. The id of a choice set defined with sf_choices().
scale_id	Character or NULL. The id of the scale this item belongs to.
reverse	Logical. Whether this item is reverse-coded within its scale.
help	Character or NULL. Help text displayed beneath the question.
placeholder	Character or NULL. Placeholder text for text inputs.
matrix_items	Character vector or NULL. Row labels for "matrix" type.
comparison_items	Character vector or NULL. The things being compared or weighted, for "pairwise_comparison" and "criteria_weight" types. At least 2 entries, all distinct. A "saaty" pairwise item renders n(n-1)/2 unordered pair rows, an "influence" item renders n(n-1) ordered rows, and a "criteria_weight" item renders one numeric input per entry. An advisory warning is raised above 7 items ("saaty") or 6 ("influence"), and above 10 the item is rejected.
comparison_scale	Character or NULL. For "pairwise_comparison" only. "saaty" (the default) gives the bipolar 1-9 importance scale used by AHP and ANP, while

	"influence" gives the unipolar 0-4 directed influence scale used by DEMATEL.
slider_min	Numeric or NULL. Minimum value for "slider" type.
slider_max	Numeric or NULL. Maximum value for "slider" type.
slider_step	Numeric or NULL. Step size for "slider" type.
rating_max	Integer or NULL. Maximum rating for "rating" type.
rating_icon	Character or NULL. Icon type: "star" or "heart".
date_min	Character or Date or NULL. Earliest selectable date for "date" type, as "YYYY-MM-DD".
date_max	Character or Date or NULL. Latest selectable date for "date" type, as "YYYY-MM-DD".
section_intro	Character or NULL. Intro text for "section_break" type.
page	Integer or NULL. Page number for multi-page surveys.

Value

An object of class sf_item (a named list).

See Also

sf_instrument(), sf_choices(), sf_scale()

Examples

```
item <- sf_item(
  id = "sat_overall", label = "Overall, how satisfied are you?",
  type = "likert", required = TRUE, choice_set = "agree5",
  scale_id = "satisfaction"
)

sec <- sf_item("sec_1", "Demographic Information", type = "section_break",
               section_intro = "Please answer the following questions.")
```

---

sf_items *Get survey items*

Description

Returns the declared question items in instrument order.

Usage

```
sf_items(x, ...)
```

Arguments

x	A surveyframe object.
...	Passed to methods.

Value

An sf_component_list for an instrument or an item data frame for a codebook.

See Also

sf_accessors

Examples

```
sf_items(sframe_demo_data()$instrument)
```

---

sf_label *Get an instrument component label*

Description

Reads the respondent- or analyst-facing label attached to a component.

Usage

```
sf_label(x, ...)
```

Arguments

x	An `sf_item()`, `sf_choices()`, `sf_scale()`, `sf_branch()`, `sf_check()` or `sf_model()` object.
...	Passed to methods.

Value

A single character label, or `""` when none is declared.

See Also

sf_identity, `sf_id()`

Examples

```
sf_label(sf_item("q1", "Satisfaction", type = "numeric"))
```

sf_meta *Get survey metadata*

Description

Reads the title, version, description, language, validation state, and other metadata without depending on the object's internal list layout.

Usage

```
sf_meta(x, ...)
```

Arguments

x	A surveyframe object.
...	Passed to methods.

Value

A list of instrument or codebook metadata.

See Also

sf_accessors

Examples

```
sf_meta(sframe_demo_data()$instrument)
```

sf_missing_columns *Columns the instrument declares that the responses left out*

Description

A partial export gives a quality report every column it holds, and none of the ones it dropped. This names the declared columns that never arrived, so a missingness figure can be read against what was expected. They count as missing for every respondent in `quality_report()`'s rates.

Usage

```
sf_missing_columns(x, ...)
```

Arguments

x	An `sframe_quality_report` object.
...	Passed to methods.

Value

A character vector of column names, empty where the export carried every declared column.

See Also

quality_report(), sf_flagged()

Examples

```
demo <- sframe_demo_data()
qr   <- quality_report(demo$responses, demo$instrument)
sf_missing_columns(qr)
```

---

sf_model *Create a surveyframe model specification*

Description

Create a surveyframe model specification

Usage

```
sf_model(
  id,
  label = NULL,
  type = c("efa", "cfa", "cb_sem", "pls_sem"),
  engine = NULL,
  constructs = list(),
  paths = list(),
  covariances = list(),
  indirect = list(),
  options = list()
)
```

Arguments

id	Model identifier.
label	Human-readable model label.
type	Model type. One of "efa", "cfa", "cb_sem", or "pls_sem".
engine	Optional engine name. Defaults to "lavaan" for CFA/CB-SEM and "seminr" for PLS-SEM.
constructs	List of sf_construct() objects.
paths	List of sf_path() objects.
covariances	List of sf_covariance() objects.
indirect	List of sf_indirect() objects.
options	List of model options, such as estimator, missing, bootstrap, or standardised.

Value

An object of class `sf_model`.

Examples

```
m <- sf_model(
  "cb1", "Quality drives intention", type = "cb_sem",
  constructs = list(
    sf_construct("sq", "Service Quality", items = c("sq_1", "sq_2", "sq_3")),
    sf_construct("sat", "Satisfaction", items = c("sat_1", "sat_2")),
    sf_construct("bi", "Behavioural Intention", items = c("bi_1", "bi_2"))
  ),
  paths = list(sf_path("sq", "sat"), sf_path("sat", "bi")),
  indirect = list(sf_indirect("sq", through = "sat", to = "bi"))
)
m$type
m$engine
```

`sf_models` *Get model specifications*

Description

Returns declared CFA, SEM, PLS-SEM, mediation, and related model objects.

Usage

```
sf_models(x, ...)
```

Arguments

`x`	A surveyframe object.
`...`	Passed to methods.

Value

An sf_component_list for an instrument or a model data frame for a codebook.

See Also

sf_accessors

Examples

```
sf_models(sframe_demo_data()$instrument)
```

sf_object *Recover the validated object*

Description

Returns the original object carried by a validation result, whether or not validation passed.

Usage

```
sf_object(x, ...)
```

Arguments

x	An sframe_validation object.
...	Passed to methods.

Value

The object held by a validation result.

See Also

sf_validation_accessors, `as_sframe()`

Examples

```
v <- validate_sframe(sframe_demo_data()$instrument, strict = FALSE)
sf_object(v)
```

sf_path *Define a structural path between constructs*

Description

Define a structural path between constructs

Usage

```
sf_path(from, to, label = NULL)
```

Arguments

from	Source construct ID.
to	Target construct ID.
label	Optional lavaan label for the path.

Value

An object of class `sf_path`.

Examples

```
p <- sf_path("sq", "sat", label = "H1")
p$from
p$to
```

`sf_plan` *Get the pre-declared analysis plan*

Description

Returns the ordered analysis blocks attached to an instrument or codebook.

Usage

```
sf_plan(x, ...)
```

Arguments

`x`	A surveyframe object.
`...`	Passed to methods.

Value

A list of analysis blocks for an instrument or a plan data frame for a codebook.

See Also

sf_accessors, sf_plan<-

Examples

```
sf_plan(sframe_demo_data()$instrument)
```

sf_plan<- *Set the pre-declared analysis plan*

Description

The replacement counterpart to `sf_plan()`. Declaring the plan is the step the whole workflow turns on, so it has a named function rather than assignment into the object's internals.

Usage

```
sf_plan(x) <- value
```

Arguments

x	An `sframe` object.
value	A list of analysis blocks.

Value

The updated `sframe` object.

See Also

`sf_plan()`, `validate_sframe()`, `run_analysis_plan()`

Examples

```
item  <- sf_item("q1", "How satisfied are you?", type = "numeric")
instr <- sf_instrument("Demo", components = list(item))

sf_plan(instr) <- list(
  list(id = "RQ1", research_question = "What is the average?",
       family = "descriptive", method = "descriptives",
       roles = list(variables = "q1"))
)
length(sf_plan(instr))
```

sf_problems *Get validation problems*

Description

Returns every actionable validation message in check order.

Usage

```
sf_problems(x, ...)
```

Arguments

x	An sframe_validation object.
...	Passed to methods.

Value

A character vector, empty when validation passed.

See Also

sf_validation_accessors, `validate_sframe()`

Examples

```
v <- validate_sframe(sframe_demo_data()$instrument, strict = FALSE)
sf_problems(v)
```

---

sf_reference *Create a reference*

Description

Builds one bibliographic reference as a CSL-JSON record, the format that Zotero, Mendeley and pandoc share. Attach references to an instrument with `add_references()` or through `sf_instrument(components = )`, cite them in text as `[@id]`, and `render_report()` lists every cited reference once, at the end of the report.

Usage

```
sf_reference(
  id,
  title,
  author = NULL,
  year = NULL,
  type = "article-journal",
  container = NULL,
  volume = NULL,
  issue = NULL,
  pages = NULL,
  publisher = NULL,
  edition = NULL,
  editor = NULL,
  doi = NULL,
  url = NULL,
  version = NULL,
  language = NULL,
  cited_by = NULL
)
```

Arguments

id	Character. The citation key, for example "cohen_1988". Starts with a letter, digit or underscore, as pandoc requires.
title	Character. The work's title.
author	Character vector of names written "Family, Given", or a single organisation name written "{R Core Team}" in braces. NULL when the work has no named author.
year	Integer or character. Year of publication.
type	Character. A CSL item type: "article-journal" (default), "book", "chapter", "paper-conference", "report", "thesis", "webpage", "software", "dataset", "manuscript" or "document".
container	Character. The journal, or the book a chapter is in.
volume, issue, pages	Character. Volume, issue and page range, for example "591-611", or an article number.
publisher	Character. Publisher, or for a thesis the institution.
edition	Character. Edition, for example "2".
editor	Character vector of editor names, written as for author.
doi	Character. The bare DOI, leaving off the https://doi.org/ prefix.
url	Character. A web address.
version	Character. A software version.
language	Character. The work's language as a code, for example "fr". Set it for a work in another language: a citation style title-cases English titles only, and would otherwise capitalise "et" or "ir" in a French or Lithuanian journal name.
cited_by	Character vector. Where the instrument cites this reference: "instrument", "item:<id>", "scale:<id>", "plan:<block id>" or "model:<id>". Optional.

Value

An object of class sf_reference.

See Also

read_bibtex(), add_references(), sframe_references()

Examples

```
ref <- sf_reference(
  id = "cronbach_1951", type = "article-journal",
  author = "Cronbach, L. J.", year = 1951,
  title = "Coefficient alpha and the internal structure of tests",
  container = "Psychometrika", volume = "16", issue = "3",
  pages = "297-334", cited_by = "scale:satisfaction"
)
ref$id
```

sf_report_accessors *Read the reportable parts of an analysis or quality result*

Description

`sf_apa()` returns the APA-formatted sentence a result carries. `sf_flagged()` returns the rows a quality report flagged.

Details

Given analysis results, `sf_apa()` answers for every block at once, as a character vector named by block. Given one of the standalone reports that carry a sentence of their own, an assumption report, a descriptives report, a missing-data report or a validity report, it returns that single sentence. A result with no sentence gives an empty string, so the shape of the answer follows the number of blocks asked about.

The sentence is plain text. APA 7 asks for italic Latin statistical symbols, so a manuscript needs `t`, `F`, `p`, `r`, `d` and the rest italicised after pasting: a character vector cannot carry that styling. Numbers are already APA-formatted, including no leading zero on `p` and a labelled confidence interval.

`sf_flagged()` returns row positions in the response data, as one sorted vector with each row once, pooling every check the quality report ran: failed attention checks, straight-lining, excess missingness, timing and duplicates. Read the report itself for which check flagged a row.

Value

`sf_apa()` returns a character vector: one element per block, named by block, for analysis results, and one element for a single report. `sf_flagged()` returns an integer vector of row positions, sorted, each row once.

Examples

```
demo <- sframe_demo_data()
qr <- quality_report(demo$responses, demo$instrument)
head(sf_flagged(qr))
```

sf_scale *Define a scored scale*

Description

Creates a scale definition that groups items and specifies how composite scores are computed. The scale carries scoring rules used by `score_scales()` and measurement structure used by `reliability_report()`, `item_report()`, and `cfa_syntax()`.

Usage

```
sf_scale(
  id,
  label,
  items,
  method = c("mean", "sum"),
  min_valid = NULL,
  reverse_items = NULL,
  weights = NULL
)
```

Arguments

id	Character. A unique identifier for this scale. Referenced in the scale_id argument of sf_item().
label	Character. A human-readable name for the scale, used in reports and codebooks.
items	Character vector. The id values of items that belong to this scale, each listed once. Order controls presentation in reports, while scoring uses the same item IDs regardless of order. Items themselves are passed to sf_instrument() as separate components.
method	Character. Scoring method. Either "mean" (default) or "sum".
min_valid	Integer or NULL. The minimum number of answered items required to compute a score for a respondent, a whole number from 1 to the number of items. When NULL, every item must be answered. An item whose column is absent from the data counts as unanswered. Used by score_scales().
reverse_items	Character vector or NULL. A subset of items that this scale reverse-codes. Reversal applies within this scale only, so the same item can be reversed in one scale and scored as answered in another. An item can also be flagged with reverse = TRUE in sf_item(), which reverses it within the scale named by its scale_id. A reversed item needs declared response bounds, from a numeric choice set, slider limits or a rating maximum.
weights	Numeric vector or NULL. Item weights for weighted scoring, one positive finite number per item, in the order of items. score_scales() applies the weights to either method = "mean" or method = "sum".

Value

An object of class sf_scale (a named list).

See Also

sf_item(), score_scales(), reliability_report()

Examples

```
sat_scale <- sf_scale(
  id           = "satisfaction",
```

```
  label         = "Customer Satisfaction",
  items         = c("sat_overall", "sat_speed", "sat_quality"),
  method        = "mean",
  min_valid     = 2,
  reverse_items = NULL
)
```

`sf_scales` *Get survey scales*

Description

Returns the scale definitions, including their item membership and scoring settings.

Usage

```
sf_scales(x, ...)
```

Arguments

`x`	A surveyframe object.
`...`	Passed to methods.

Value

An sf_component_list for an instrument or a scale data frame for a codebook.

See Also

sf_accessors

Examples

```
sf_scales(sframe_demo_data()$instrument)
```

`sf_validation_accessors`
Read a validation diagnostic

Description

`sf_is_valid()` reports whether the object passed. `sf_problems()` returns the problem messages. `sf_object()` returns the object that was validated.

Value

`sf_is_valid()` returns a single logical. `sf_problems()` returns a character vector, empty when the object is valid. `sf_object()` returns the validated object.

See Also

`validate_sframe()`, sframe_validation, `as_sframe()`

Examples

```
item  <- sf_item("q1", "How satisfied are you?", type = "text")
instr <- sf_instrument("Demo", components = list(item))
v <- validate_sframe(instr, strict = FALSE)

sf_is_valid(v)
sf_problems(v)
```

---

sframe_aggregate_judgements

Aggregate individual judgement matrices

Description

Aggregation of individual judgements (AIJ) across respondents. The element-wise geometric mean is the standard choice for reciprocal AHP matrices, because it is the only mean that preserves reciprocity: the arithmetic mean of `m[a, b]` and the arithmetic mean of `m[b, a]` are not reciprocals of each other. DEMATEL matrices are not reciprocal, so they aggregate arithmetically.

Usage

```
sframe_aggregate_judgements(
  matrices,
  method = c("geometric", "arithmetic"),
  cr_filter = FALSE
)
```

Arguments

`matrices`	Either a list of square numeric matrices or the object returned by `sframe_assemble_pairwise()`.
`method`	`"geometric"` (the AHP default) or `"arithmetic"` (DEMATEL).
`cr_filter`	Logical. When `TRUE`, individual matrices with a consistency ratio at or above 0.10 are dropped before aggregation. Applies to reciprocal matrices only. Default `FALSE`.

Value

A list with `matrix`, `method`, `n_respondents`, `n_dropped`, `n_dropped_consistency`, and `consistency` (the per-respondent CR distribution, or `NULL` for a non-reciprocal set).

See Also

`sframe_assemble_pairwise()`

Examples

```
m1 <- matrix(c(1, 3, 1 / 3, 1), 2, 2)
m2 <- matrix(c(1, 5, 1 / 5, 1), 2, 2)
sframe_aggregate_judgements(list(m1, m2))$matrix
```

---

sframe_analysis_qmd *Write a Quarto analysis notebook for any instrument*

Description

Unlike `sframe_demo_qmd()`, which only works for one of the bundled demo instruments (it looks `name` up in `sframe_demos()`), this writes a runnable Quarto notebook for any instrument, using its own responses. It exposes the complete ordered plan, screening, run status, detailed results, plots and reports. The notebook reads the instrument and its responses back from 2 companion files written alongside it, so all 3 files must stay together.

Usage

```
sframe_analysis_qmd(
  instrument,
  data,
  dir = ".",
  basename = NULL,
  overwrite = FALSE
)
```

Arguments

`instrument`	An `sframe` object.
`data`	A `data.frame` of responses, read by `read_responses()` when the notebook runs.
`dir`	Directory to write into. Defaults to the working directory.
`basename`	Character or `NULL`. File base name shared by the `.qmd`, `.sframe`, and `_responses.csv` files. Defaults to a slug of the instrument's title.
`overwrite`	Logical. Overwrite existing files of the same name.

Value

A list with qmd, sframe, and csv paths, invisibly.

See Also

sframe_demo_qmd(), write_sframe(), read_responses()

Examples

```
item  <- sf_item("q1", "How satisfied are you?", type = "text")
instr <- sf_instrument("Demo", components = list(item))
resp  <- data.frame(q1 = c("Great", "Fine"))
out <- sframe_analysis_qmd(instr, resp, dir = tempdir())
basename(out$qmd)
```

---

sframe_as_data_frame *Coerce a surveyframe object to a data frame*

Description

Every surveyframe class returns its primary table, and each class has its own columns. Where an object holds more than one table, the others are reachable through the named accessors in sf_accessors, or, for a full tabular record of an instrument, through codebook_report().

Value

A data frame, with the columns listed above for the class given.

What each class gives

- sframe: one row per item, with id, label, type, choice_set, scale_id, reverse and required.
- sframe_codebook: one row per item, the codebook's item table.
- sframe_validation: one row per problem, with check and problem.
- sframe_analysis_results: one row per block, with block, research_question, method and apa.
- sframe_reliability_report: one row per scale, with scale_id, label, n_items, n, alpha and omega.
- sframe_item_report: one row per item, with scale_id and the item diagnostics.
- sframe_quality_report: one row per check, the flattened quality checks.
- sframe_efa_report: one row per measure, the readiness measures.
- sframe_sensitivity: one row per perturbation, with criterion, direction, weight, rho, rank_changed and top_changed.

A summary, and where the full record is

These tables are a summary of the columns a reader scans first. The item table leaves out help text, placeholder, matrix rows, comparison items and scale, slider and rating settings, date bounds, section introduction and page; the scale table leaves out `min_valid`, the reverse key and the weights. Read the stored declaration in full through `sf_items()`, `sf_scales()` and the rest of sf_accessors, each of which returns the component objects themselves, or through `write_sframe()` for the interchange record.

A class holding one table returns it directly, so it keeps that table's own row names and `row.names` has no effect. Pass `row.names` to `base::as.data.frame()` on the returned frame where you need to set them. The coercion gives one view of an object. An instrument, for example, returns its items, and its choice sets and scales come from `sf_choice_sets()`, `sf_scales()` or `codebook_report()`.

See Also

sf_accessors, `codebook_report()`

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
item  <- sf_item("sat_1", "The service met my expectations.",
                 type = "likert", choice_set = "ag5", scale_id = "sat")
scale <- sf_scale("sat", "Satisfaction", items = "sat_1")
instr <- sf_instrument("Demo Survey", components = list(cs, item, scale))

as.data.frame(instr)
as.data.frame(cs)
```

---

sframe_assemble_pairwise

Assemble per-respondent comparison matrices

Description

Builds one square judgement matrix per respondent from the pair columns of a `"pairwise_comparison"` item. A `"saaty"` item stores one signed integer per unordered pair, so the reciprocal half of each matrix is reconstructed here rather than stored (a collected value of +5 becomes `m[a, b] = 5` and `m[b, a] = 1/5`, and the diagonal is 1). An `"influence"` item stores one unsigned integer per ordered pair, fills the directed cell only, and has a zero diagonal, because the influence of a on b says nothing about the influence of b on a.

Usage

```
sframe_assemble_pairwise(data, instrument, item_id)
```

Arguments

data	A data frame of responses, as produced by read_responses().
instrument	An sframe instrument declaring item_id.
item_id	Character. The id of a "pairwise_comparison" item.

Details

Respondents who left any pair blank, or whose answer falls outside the declared scale, are dropped whole and counted. Completing partial matrices (the Harker approach) is deliberately out of scope.

Value

A list with matrices (a list of square numeric matrices with dimnames), n_respondents, n_dropped, dropped (a data frame of row number and reason), items, and scale.

See Also

sframe_aggregate_judgements(), sframe_collected_weights()

Examples

```
crits <- c("price", "speed")
study <- sf_instrument(
  title = "Demo", components = list(
    sf_item("pairs", "Compare the criteria", type = "pairwise_comparison",
            comparison_items = crits)
  )
)
responses <- data.frame(pairs__price__vs__speed = c(3, -5))
sframe_assemble_pairwise(responses, study, "pairs")$matrices[[1]]
```

---

sframe_bibliography *Write a bibliography as CSL-JSON*

Description

Writes the references a report cites as CSL-JSON, which pandoc, Quarto and reference managers read directly. It holds the instrument's own references, the method references of any analysis results given, and any further references a host supplies, each once. render_report() uses this for its References section, and a host that builds its own documents can use the same file.

Usage

```
sframe_bibliography(
  instrument = NULL,
  results = NULL,
  extra = NULL,
  file = NULL
)
```

Arguments

instrument	An sframe object, or NULL.
results	Optional results from run_analysis_plan(), whose methods' references are included.
extra	Optional further sf_reference objects, for example the references a host keeps for the write-up.
file	Path to write. When NULL, the CSL-JSON is returned as a string.

Value

The path, invisibly, or the JSON string when file is NULL.

Examples

```
instr <- sframe_demo("two_group")$instrument
cat(substr(sframe_bibliography(instr), 1, 200))
```

---

sframe_builder_empty_state

Create an empty SurveyStudio builder state

Description

Create an empty SurveyStudio builder state

Usage

```
sframe_builder_empty_state()
```

Value

A list containing empty metadata, choice, item, scale, branching, and check collections suitable for SurveyStudio.

Examples

```
state <- sframe_builder_empty_state()
state$meta$title
length(state$items)
```

sframe_builder_state_from_instrument

Convert an instrument into a SurveyStudio builder state

Description

Convert an instrument into a SurveyStudio builder state

Usage

```
sframe_builder_state_from_instrument(instrument = NULL)
```

Arguments

instrument	An sframe object or NULL.

Value

A builder state list. Component classes are restored so the state can be edited or validated by SurveyStudio.

Examples

```
demo <- sframe_demo_data()
state <- sframe_builder_state_from_instrument(demo$instrument)
length(state$items)
length(state$scales)
```

sframe_builder_validate_draft

Validate a SurveyStudio draft state

Description

Validate a SurveyStudio draft state

Usage

```
sframe_builder_validate_draft(
  meta,
  choices = list(),
  items = list(),
  scales = list(),
  branching = list(),
  checks = list(),
  analysis_plan = list(),
```

```
  models = list(),
  render = list(),
  amendments = list(),
  designs = list(),
  references = list(),
  origin = NULL
)
```

Arguments

meta	List of instrument metadata.
choices, items, scales, branching, checks	Lists of draft components.
analysis_plan	List of draft analysis-plan blocks.
models	List of draft model specifications.
render	List of rendering settings (welcome, header/logo, thankyou, theme) carried from the loaded instrument so previews and exports match.
amendments	List of previously disclosed amendment entries, carried through unchanged so a draft round trip does not drop them.
designs	List of conjoint designs, carried through unchanged, since Studio has no editor for them.
references	List of sf_reference() records, as edited on Studio's References screen.
origin	The load record of an instrument read with read_sframe(), from the builder state, or NULL. Carried onto the draft so an edit of a loaded file is recognised as a revision.

Value

A list with valid, problems, instrument, and revision_problem: the reason write_sframe() would refuse the draft as an undisclosed revision, or NULL.

Examples

```
demo  <- sframe_demo_data()
state <- sframe_builder_state_from_instrument(demo$instrument)
draft <- sframe_builder_validate_draft(
  meta = state$meta, choices = state$choices, items = state$items,
  scales = state$scales, branching = state$branching, checks = state$checks
)
draft$valid
```

sframe_codebook_items_display

Enrich a codebook's items table for display

Description

Replaces `items_table`'s `choice_set` id with the choice set's actual response options ("1 = Strongly disagree; 2 = Disagree; ...") and its `scale_id` with the scale's label, so each row of the printed codebook is self-contained. `codebook_report()` itself keeps the raw ids, for joining `items_table` to `choices_table`/`scales_table` programmatically. This enrichment is for the rendered document, where a reader should not need to cross-reference a separate table just to see what "1" means on a scale shared by many items.

Usage

```
sframe_codebook_items_display(cb)
```

Arguments

cb	An `sframe_codebook` object from `codebook_report()`.

Value

A data.frame, `cb$items_table` with `choice_set` and `scale_id` replaced by display text.

See Also

`codebook_report()`

Examples

```
demo <- sframe_demo_data()
cb <- codebook_report(demo$instrument)
head(sframe_codebook_items_display(cb))
```

sframe_collected_weights

Criterion weights collected from respondents

Description

Resolves a weight vector from either kind of collected weight item, so that a ranking runner never has to know which one the researcher used. A `"criteria_weight"` item is renormalised to sum 1 per respondent before the arithmetic mean is taken, so a respondent who allocated 90 points in total carries the same influence as one who allocated exactly 100. A `"pairwise_comparison"` item is assembled, aggregated geometrically, and reduced to its principal eigenvector.

Usage

```
sframe_collected_weights(data, instrument, item_id, cr_filter = FALSE)
```

Arguments

data	A data frame of responses.
instrument	An sframe instrument declaring item_id.
item_id	Character. A "criteria_weight" or "pairwise_comparison" item id.
cr_filter	Logical. Passed to sframe_aggregate_judgements() for a pairwise item. Ignored otherwise.

Value

A list with weights (a named numeric vector summing to 1), criteria, source, item_id, n_respondents, n_dropped, and consistency (pairwise items only).

See Also

sframe_assemble_pairwise()

Examples

```
study <- sf_instrument(
  title = "Demo", components = list(
    sf_item("pts", "Divide 100 points", type = "criteria_weight",
            comparison_items = c("price", "speed"))
  )
)
sframe_collected_weights(
  data.frame(pts__price = c(60, 40), pts__speed = c(40, 60)), study, "pts"
)$weights
```

---

sframe_decision_options

Normalise the decision options of an analysis-plan block

Description

A researcher-supplied performance matrix round-trips through JSON as a list of numeric row vectors with its dimnames dropped, so it is stored as options$matrix plus the options$alternatives and options$criteria label vectors and rebuilt here. Every length agreement between the matrix, its labels, the weights, and the criterion types is checked once, in one place, with the exact mismatch named.

Usage

```
sframe_decision_options(options)
```

Arguments

options	The options list of a decision analysis block.

Value

The same list with matrix rebuilt as a numeric matrix carrying dimnames, and weights and criteria_types coerced and checked.

PROMETHEE preference functions

A PROMETHEE block takes options$preference_function, one of "usual", "linear", or "level", and options$thresholds for the 2 that need them. The default is "usual", Brans and Vincke's type I step function, which needs no thresholds and so adds no researcher degrees of freedom.

This default differs from several other MCDM implementations, which default to the linear (V-shape) function and derive its thresholds from the range of the supplied data. Deriving thresholds that way makes the result depend on choices the researcher never declared, which is what this package exists to prevent, so surveyframe requires a threshold-bearing preference function to be asked for explicitly.

The choice changes the answer. Net flows always differ between the 2 functions, and the ranking itself changed in 226 of 400 randomly drawn 4-alternative by 3-criterion matrices. So a ranking cross-checked against an implementation that defaults to "linear" will often disagree unless preference_function = "linear" is set here and the same thresholds are supplied on both sides. The preference function actually used is always named in the block's APA sentence. Note also that "usual" produces tied ranks readily, because a step function scores every non-zero difference identically.

See Also

run_analysis_plan()

Examples

```
sframe_decision_options(list(
  matrix = list(c(4, 210), c(3, 180)),
  alternatives = c("Alpha", "Basilica"),
  criteria = c("service", "price"),
  criteria_types = c("benefit", "cost")
))$matrix
```

---

sframe_dematel_compute

The DEMATEL total-relation classification

Description

Normalises the direct-influence matrix `X` by the larger of its greatest row sum and its greatest column sum (the standard DEMATEL normalisation, which keeps the Neumann series `N + N^2 + N^3 + ...` convergent), then solves the total-relation matrix `T = N(I - N)^-1` in closed form rather than by truncating the series. `D` is each criterion's row sum of `T` (how much it influences the others, direct and indirect combined) and `R` its column sum (how much it is influenced). Prominence `D + R` is overall involvement in the system, while relation `D - R` is net direction, positive for a net cause and negative or zero for a net effect. The threshold is the arithmetic mean of every entry of `T`: relations at or above it are considered significant enough to draw in an influence diagram.

Usage

```
sframe_dematel_compute(x)
```

Arguments

`x`	A square numeric matrix of direct influence, zero diagonal.

Details

The series converges only when the spectral radius of `N` is below 1. Normalisation keeps the radius at or below 1, and it equals 1 when criteria influence each other in a closed group with equal totals. That case returns an `error` explaining it, and no truncated series is substituted.

Value

A list with `normalised` (N), `total_relation` (T), `D`, `R`, `prominence` (D + R), `relation` (D - R), `threshold` (mean of T), and `role` (a character vector, `"cause"` where relation > 0, else `"effect"`).

`sframe_demo` *Load one bundled demo*

Description

One of the 22-item teaching library: 17 analysis examples, three presentation examples, and two provenance examples. For the demo that instead exercises every input type SurveyBuilder and SurveyStudio support in a single instrument, see `sframe_input_types_demo_data()`.

Usage

```
sframe_demo(name, branded = FALSE)
```

Arguments

`name`	Character. A demo name, as listed by `sframe_demos()`.
`branded`	Logical. When `TRUE`, the instrument comes back with the standard welcome page, logo, theme colour and thank you page spliced into its `render` block. The bundled file on disk is left unchanged, so the same branding can be shown on whichever demo matches your own survey. See `sframe_demo_branding()`.

Value

A list with `instrument`, `responses`, and the paths behind them: `instrument_path`, `responses_path`, `codebook_path` and `results_path`. The codebook carries variable and value labels, so the data means something outside R. The results table is what surveyframe reports for this demo, which is the reference to compare against when you run the same data through another package.

See Also

`sframe_demos()`, `sframe_export_labelled()`, `sframe_input_types_demo_data()`

Examples

```
demo <- sframe_demo("two_group")
demo$instrument
head(demo$responses)
```

`sframe_demo_branding` *The standard demo branding*

Description

The `render` block `sframe_demo()` splices in when `branded = TRUE`. Read it, change a colour, and paste it into your own instrument.

Usage

```
sframe_demo_branding()
```

Value

A named list of class `sframe_branding`, suitable for `sf_instrument(render = )`: the display `mode`, the `theme` colour, the `submit_label`, and the `welcome`, `thankyou` and `header` blocks. Printing it shows each setting as a respondent meets it. `sf_instrument()` and `sframe_demo()` store it as a plain list.

See Also

`preview_demo()` to see the branding applied to a demo.

Examples

```
branding <- sframe_demo_branding()
branding
branding$welcome$title
branding$theme
```

sframe_demo_data *Load bundled surveyframe demo data*

Description

Loads the bundled tourism-services `.sframe` instrument and simulated response dataset used in package examples and statistical workflow demos.

Usage

```
sframe_demo_data()
```

Value

A list with `instrument`, `responses`, `instrument_path`, and `responses_path`.

Examples

```
demo <- sframe_demo_data()
sf_meta(demo$instrument)$title
nrow(demo$responses)
```

sframe_demo_qmd *Copy a demo's Quarto notebook so you can run and change it*

Description

The code route through a demo is a notebook that renders to a report, which is what a research workflow looks like. This writes one for the named demo: load and validate the instrument and responses, inspect the complete ordered analysis plan, screen data quality, run the plan with status and plots, inspect each result and its reproducible syntax, compare the bundled expected results, render analysis and full reports, and export the data for checking elsewhere.

Usage

```
sframe_demo_qmd(name, dir = ".", overwrite = FALSE)
```

Arguments

`name`	Character. A demo name, as listed by `sframe_demos()`.
`dir`	Directory to write into. Defaults to the working directory.
`overwrite`	Logical. Overwrite an existing file of the same name.

Value

The path written, invisibly.

See Also

sframe_demo(), sframe_demos()

Examples

```
out <- sframe_demo_qmd("two_group", dir = tempdir())
basename(out)
```

---

sframe_demos *List the bundled demos*

Description

Every demo does one job, so a failure points at one method and a reader can hold the whole questionnaire in view. Use sframe_demo() to load one.

Usage

```
sframe_demos()
```

Value

A data frame with 1 row per demo: its name, whether its focus is analysis, presentation or provenance, what it teaches, the input fields it uses, the statistical technique it demonstrates, and the demo whose responses it reuses, if any.

See Also

sframe_demo(), sframe_demo_branding(), sframe_demo_qmd()

Examples

```
head(sframe_demos(), 3)
subset(sframe_demos(), focus == "provenance")
```

`sframe_export_labelled` *Write responses to SPSS or Stata with the labels attached*

Description

A plain CSV carries codes. `dm_1` arrives in SPSS as a column of integers with no variable label and no value labels, and the reader has to reconstruct all of it from the questionnaire. This attaches both from the instrument: the variable label is the item's `label`, and the value labels come from the item's choice set, which stores its `values` and `labels` side by side.

Usage

```
sframe_export_labelled(data, instrument, path)
```

Arguments

`data`	A response data frame, as returned by `read_responses()`.
`instrument`	The `sframe` the responses were collected with.
`path`	Output file. An `.sav` is written for SPSS, a `.dta` for Stata, chosen from the extension.

Value

The path, invisibly. When any name or label was changed, the path carries an attribute `"renamed"`, a data frame with the columns `kind` (`"name"`, `"variable label"` or `"value label"`), `column`, `old` and `new`. With no changes there is no message and no attribute.

Text codes in Stata

Stata attaches value labels to numbers alone. In a `.dta`, a choice set coded with text, such as `"a"` and `"b"`, is written as 1, 2, 3 in the set's declared order, each carrying its label, so the file reads back in Stata with the labels intact. A `.sav` keeps the text codes, which SPSS labels directly.

Names and labels that do not fit the format

SPSS and Stata refuse column names with spaces or punctuation, names that do not start with a letter, and names over the format's length limit (64 bytes for SPSS, 32 characters for Stata). Matrix row columns such as `session__Opening keynote` are renamed to valid names, with illegal characters turned into `_`, a leading `v` where the first character is not a letter, and a number added where two names would otherwise match. Variable labels are cut at 256 bytes for SPSS and 80 characters for Stata, and value labels at 120 bytes for SPSS and 80 characters for Stata. Everything else keeps its full label. One message lists each change as old -> new.

See Also

`sframe_demo()`, `read_responses()`

Examples

```
demo <- sframe_demo("two_group")
out <- file.path(tempdir(), "two_group.sav")
if (requireNamespace("haven", quietly = TRUE)) {
  sframe_export_labelled(demo$responses, demo$instrument, out)
}
```

`sframe_input_types_demo_data` *Load the input-types demo backing SurveyBuilder and SurveyStudio*

Description

Loads the bundled `.sframe` instrument and simulated response dataset that cover all main survey input types supported by surveyframe. This is a different demo from the 22-item teaching library behind `sframe_demo()`: where `sframe_demo()` demonstrates one analysis method per call, this one exists to exercise every input control SurveyBuilder and SurveyStudio support in a single instrument, and is what backs `launch_builder_demo()`, `launch_studio_demo()`, and `launch_dashboard_demo()`.

Usage

```
sframe_input_types_demo_data()
```

Value

A list with `instrument`, `responses`, `instrument_path`, and `responses_path`.

See Also

`sframe_demo()` for the 22-item teaching library instead.

Examples

```
demo <- sframe_input_types_demo_data()
sf_meta(demo$instrument)$title
nrow(demo$responses)
```

sframe_likert_scale_groups

Group a scale's Likert items for a combined diverging chart

Description

Identifies which of an instrument's scales are eligible for one grouped diverging chart across their member items (sframe_plot_likert_scale()), the same way a "matrix" question's rows are grouped (sframe_plot_likert_matrix()): every member item is "likert" type and all share one choice set. Scales that mix response scales, that resolve to fewer than 2 qualifying items, or whose choice set cannot be found are left out and fall back to one chart per item in the report's Response distributions section.

Usage

```
sframe_likert_scale_groups(instrument)
```

Arguments

instrument	An sframe object.

Value

A named list, one entry per eligible scale (named by scale id), each a list with scale_id, title (the scale's label), items (the member item objects, in scale order), and choice_set (the shared choice set object). Empty list if no scale qualifies.

See Also

sframe_plot_likert_scale(), sf_scale()

Examples

```
demo <- sframe_demo("likert_scale")
groups <- sframe_likert_scale_groups(demo$instrument)
names(groups)
```

sframe_plot_cooccurrence

Term co-occurrence heatmap

Description

Tile heatmap of pairwise within-response term co-occurrence counts for a co_occurrence result. The result's edge list (term_a, term_b, n) is pivoted into a full symmetric term-by-term grid before plotting, so each pair's tile appears twice, once on either side of the diagonal, the way the other tile heatmaps in this file (sframe_plot_correlation_matrix(), sframe_plot_efa_loadings()) read as a full grid rather than a triangle.

Usage

```
sframe_plot_cooccurrence(result, palette = c("web", "print"))
```

Arguments

result	A co_occurrence result list from run_analysis_plan().
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object, or NULL when the result carries no table.

See Also

run_analysis_plan(), term_frequency()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("open_text")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_cooccurrence(res$RQ4)
}
```

---

sframe_plot_cooccurrence_network

Term co-occurrence network plot

Description

Plots a co_occurrence_network result's node table (term, frequency, cluster, x, y) as a network diagram: edges (from result$edges) as line segments underneath, nodes as points sized by term frequency and coloured by Louvain cluster, with term labels on the larger points only. Labelling every point on a dense network risks overlap chaos, so only the top 15 nodes by frequency are labelled; the full term list stays available in result$table.

Usage

```
sframe_plot_cooccurrence_network(result, palette = c("web", "print"))
```

Arguments

result	A co_occurrence_network result list from run_analysis_plan(), carrying table and edges.
palette	One of "web" or "print". See sframe_brand().

Details

Clusters beyond the first 8 (ranked largest first) are folded into a single "Other" bucket rather than cycling or interpolating a new hue, per the dataviz skill's categorical-colour guidance; see `.sframe_cluster_palette()`.

Value

A ggplot2 object, or `NULL` when the result carries no table.

See Also

`run_analysis_plan()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("igraph", quietly = TRUE)) {
  demo <- sframe_demo("open_text")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_cooccurrence_network(res$RQ6)
}
```

---

sframe_plot_correlation_matrix

Correlation matrix heatmap

Description

Computes and plots a full pairwise correlation matrix, independent of `run_analysis_plan()`'s pairwise `correlation_pearson/_spearman/ _kendall` runners (which plot one variable pair at a time via `sframe_plot_correlation()`). Useful directly, and as the visual companion to `validity_report()`'s discriminant-validity checks.

Usage

```
sframe_plot_correlation_matrix(
  data,
  vars,
  method = "pearson",
  palette = c("web", "print")
)
```

Arguments

data	A data frame of survey responses.
vars	Character vector of column names to correlate.
method	One of "pearson", "spearman", "kendall".
palette	One of "web" (diverging red/teal gradient) or "print" (white-to-black gradient by magnitude, signed label). See sframe_brand().

Value

A ggplot2 object.

See Also

validity_report()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo_data()
  sframe_plot_correlation_matrix(demo$responses,
                                 c("sq_1", "sq_2", "sq_3", "sat_1", "sat_2"))
}
```

---

sframe_plot_decision_ranking

Ranked-score bar chart for a decision-family result

Description

The shared chart for every MCDM ranking method: one horizontal bar per alternative, ordered best first, with the leading alternative picked out. It is generic over the method rather than tied to one, so AHP criterion weights and any ranking method's scores all draw through it. The score column is whatever the method reports as its headline quantity (a closeness coefficient, a net flow, a priority weight), so the axis is labelled from the result rather than hard-coded.

Usage

```
sframe_plot_decision_ranking(result, palette = c("web", "print"))
```

Arguments

result	A decision-family result list from run_analysis_plan(), carrying either scores and alternatives or a ranking table.
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object, or NULL when the result carries no ranking.

See Also

run_analysis_plan()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("mcdm_choice")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_decision_ranking(res$RQ4)
}
```

---

sframe_plot_dematel_influence

Prominence-relation scatter for a DEMATEL result

Description

The dedicated DEMATEL chart: one point per criterion, prominence (D + R) on x and relation (D - R) on y, with a horizontal quadrant line at relation = 0 separating causes (above) from effects (below). This is deliberately a different shape from sframe_plot_decision_ranking(): DEMATEL classifies criteria on two axes rather than producing a single ranked score, so a ranking bar chart would misrepresent it.

Usage

```
sframe_plot_dematel_influence(result, palette = c("web", "print"))
```

Arguments

result	A dematel-family result list from run_analysis_plan() or sframe_run_dematel(), carrying prominence, relation, and criteria.
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object, or NULL when the result carries no DEMATEL fields to plot.

See Also

sframe_plot_decision_ranking()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("mcdm_choice")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_dematel_influence(res$RQ3)
}
```

`sframe_plot_descriptives` *Distribution shape by variable, standardised*

Description

One violin per variable in a `descriptives_report()` table, built from the underlying response data rather than from the summary skewness and kurtosis numbers, so the reader sees the actual shape (asymmetry, multimodality, tails) instead of reading it off a bar height. Each variable is standardised (z-scored) before plotting so variables on different original scales (a 5-point Likert item next to a 0-100 slider) share one comparable y-axis. Standardising is a linear transform and does not change skewness. Each violin's subtitle-free panel keeps the variable's skewness value in its axis label. Grouped `descriptives_report()` output (one row per variable per `split_by` group) is faceted by group.

Usage

```
sframe_plot_descriptives(x, data, palette = c("web", "print"))
```

Arguments

`x`	An `sframe_descriptives_report` object from `descriptives_report()`.
`data`	The same data.frame passed to `descriptives_report()`. Required: `x` only carries the summary table, not the raw values the violins need.
`palette`	One of `"web"` or `"print"`. See `sframe_brand()`.

Value

A ggplot2 object, or `NULL` if none of the report's variables have enough data to draw.

See Also

`descriptives_report()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo_data()
  dr <- descriptives_report(demo$responses, variables = c("sat_1", "sat_2"))
  sframe_plot_descriptives(dr, demo$responses)
}
```

`sframe_plot_efa_loadings` *Loadings heatmap from a fitted EFA solution*

Description

Loadings heatmap from a fitted EFA solution

Usage

```
sframe_plot_efa_loadings(x, palette = c("web", "print"))
```

Arguments

`x`	An `sframe_efa_solution` object from `efa_solution()`.
`palette`	One of `"web"` (diverging red/teal gradient) or `"print"` (white-to-black gradient by magnitude, with sign conveyed by the printed label rather than colour, so it stays legible in monochrome). See `sframe_brand()`.

Value

A ggplot2 object.

See Also

`efa_solution()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("psych", quietly = TRUE)) {
  demo <- sframe_demo_data()
  fit <- efa_solution(demo$responses, demo$instrument,
                      scales = "service_quality", nfactors = 1)
  sframe_plot_efa_loadings(fit)
}
```

sframe_plot_efa_scree *Scree plot from an EFA readiness report*

Description

Plots the parallel-analysis eigenvalues from `efa_report()` (both the observed factor-analysis eigenvalues and the simulated comparison line), with the suggested factor count marked.

Usage

```
sframe_plot_efa_scree(x, palette = c("web", "print"))
```

Arguments

x	An `sframe_efa_report` object from `efa_report()`.
palette	One of `"web"` or `"print"`. See `sframe_brand()`.

Value

A ggplot2 object.

See Also

`efa_report()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("psych", quietly = TRUE)) {
  demo <- sframe_demo_data()
  er <- efa_report(demo$responses, demo$instrument)
  sframe_plot_efa_scree(er)
}
```

sframe_plot_group_comparison *Group-comparison boxplot*

Description

Boxplot with jittered points, shared across every runner whose result carries `vars = c(group_column, outcome_column)`: `t_test_ind`, `mann_whitney`, `kruskal_wallis`, and `anova_one`. One function instead of four, since the underlying comparison (an outcome split by a grouping factor) and the data shape needed to plot it are identical across all four tests, and only the inferential statistic differs.

Usage

```
sframe_plot_group_comparison(result, data, palette = c("web", "print"))
```

Arguments

result	A result list from one of the four runners above, with vars = c(group_column, outcome_column).
data	The response data frame the result was computed from.
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object, or NULL if the columns are missing, fewer than two groups remain after removing missing values, or ggplot2 is unavailable.

See Also

run_analysis_plan()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo_data()
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_group_comparison(res$rq_visit_bi, demo$responses)
}
```

---

sframe_plot_likert_matrix

Grouped diverging chart for a Likert matrix question

Description

A matrix question asks several rows against one shared response scale (a "grid" of Likert items). Plotting each row as its own separate sframe_draw_likert_diverging() chart loses the grouping the question was designed with, so this draws every row as one diverging bar inside a single chart, sharing one x scale and one legend, the standard way a Likert matrix is reported (compare a typical multi-item satisfaction grid). Same diverging-stack convention as the single-item chart: the middle category of an odd-length scale is neutral and split evenly across the zero line, and colour saturation increases toward each pole.

Usage

```
sframe_plot_likert_matrix(item, data, choice_set, palette = c("web", "print"))
```

Arguments

item	A "matrix" sframe item, with matrix_items (the row labels) and a choice_set naming the shared response scale.
data	The response data.frame, with one expanded <item id>__<row label> column per matrix row, as produced by read_responses().
choice_set	The item's choice set object (values, labels), typically looked up from instrument$choices by item$choice_set.
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object, or NULL if no row has response data.

See Also

sframe_draw_likert_diverging()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("matrix_likert")
  item <- Filter(function(i) i$type == "matrix", demo$instrument$items)[[1]]
  cs   <- Filter(function(c) c$id == item$choice_set, demo$instrument$choices)[[1]]
  sframe_plot_likert_matrix(item, demo$responses, cs)
}
```

---

sframe_plot_likert_scale

Grouped diverging chart for a scale's Likert items

Description

Several *separate* Likert items that make up one sf_scale() (unlike a "matrix" item's rows, which are one question) are, by default, each reported as their own single-item diverging chart. That scatters a related batch of items (a satisfaction scale's 2-3 items, say) across several charts instead of showing them the way a Likert matrix or a typical multi-item satisfaction grid is reported: one grouped chart, one diverging bar per item, sharing an x scale and a legend. Applies only when every item in the scale shares the same choice set. Scales that mix response scales fall back to one chart per item.

Usage

```
sframe_plot_likert_scale(
  items,
  data,
  choice_set,
  title,
  palette = c("web", "print")
)
```

Arguments

items	A list of "likert" sframe items belonging to one scale, in display order.
data	The response data.frame, with one column per item id.
choice_set	The shared choice set object (values, labels).
title	Chart title, typically the scale's label.
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object, or NULL if no item has response data.

See Also

sframe_plot_likert_matrix(), sf_scale()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("likert_scale")
  groups <- sframe_likert_scale_groups(demo$instrument)
  g <- groups[["organisation"]]
  sframe_plot_likert_scale(g$items, demo$responses, g$choice_set, g$title)
}
```

---

sframe_plot_missingness

Missing-data report plot: missingness rate by item

Description

Missing-data report plot: missingness rate by item

Usage

```
sframe_plot_missingness(x, palette = c("web", "print"))
```

Arguments

x	An sframe_missing_data_report object from missing_data_report().
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object. When no item has missing values, this is a short "no missing responses" message rather than an empty bar chart.

See Also

missing_data_report()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo_data()
  mr <- missing_data_report(demo$responses, demo$instrument)
  sframe_plot_missingness(mr)
}
```

---

sframe_plot_ngram_frequency

N-gram-frequency plot: horizontal bar

Description

Top 20 n-grams from an ngram_freq result as a horizontal bar chart. Shares its bar-building logic with sframe_plot_term_frequency()'s bar path via the internal .sframe_plot_term_bar() helper; unlike that function, there is no word-cloud mode and no group faceting for this id.

Usage

```
sframe_plot_ngram_frequency(result, palette = c("web", "print"))
```

Arguments

result	An ngram_freq result list from run_analysis_plan().
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object, or NULL when the result carries no table.

See Also

run_analysis_plan(), ngram_frequency()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("open_text")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_ngram_frequency(res$RQ2)
}
```

`sframe_plot_paired_comparison` *Paired-comparison slope plot*

Description

One line per respondent connecting their two paired values, shared by `t_test_pair` and `wilcoxon_pair` (both carry `vars = c(x_column, y_column)` on the same respondents). The standard visual for a paired design: it shows the direction and consistency of individual change, which a plain bar-of-means would hide.

Usage

```
sframe_plot_paired_comparison(result, data, palette = c("web", "print"))
```

Arguments

`result`	A result list from `t_test_pair`/`wilcoxon_pair`, with `vars = c(x_column, y_column)`.
`data`	The response data frame the result was computed from.
`palette`	One of `"web"` or `"print"`. See `sframe_brand()`.

Value

A ggplot2 object, or `NULL` if fewer than two complete pairs remain, or ggplot2 is unavailable.

See Also

`run_analysis_plan()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo_data()
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_paired_comparison(res$rq_ttest_pair, demo$responses)
}
```

sframe_plot_quality *Quality report plot: straight-lining flag rate by scale*

Description

Quality report plot: straight-lining flag rate by scale

Usage

```
sframe_plot_quality(x, palette = c("web", "print"))
```

Arguments

x	An sframe_quality_report object from quality_report().
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object.

See Also

quality_report()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("likert_scale")
  qr <- quality_report(demo$responses, demo$instrument)
  sframe_plot_quality(qr)
}
```

sframe_plot_regression_diagnostics *Regression diagnostic plots for a regression_linear result*

Description

The four standard diagnostic panels (residuals vs fitted, normal Q-Q, scale-location, residuals vs leverage), built from the plain data frame run_analysis_plan() attaches to a regression_linear result rather than the lm object itself, so the result stays JSON-serialisable.

Usage

```
sframe_plot_regression_diagnostics(result, palette = c("web", "print"))
```

Arguments

result	A regression_linear result list containing a diagnostics data frame (as produced internally by run_analysis_plan()).
palette	One of "web" or "print". See sframe_brand().

Value

A named list of four ggplot2 objects (residuals_fitted, qq, scale_location, leverage), or NULL if diagnostics are unavailable.

See Also

run_analysis_plan()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo_data()
  res <- run_analysis_plan(demo$responses, demo$instrument)
  panels <- sframe_plot_regression_diagnostics(res$rq_predict_sat)
  panels$residuals_fitted
}
```

---

sframe_plot_reliability

Reliability plot: alpha and omega by scale

Description

Reliability plot: alpha and omega by scale

Usage

```
sframe_plot_reliability(x, palette = c("web", "print"))
```

Arguments

x	An sframe_reliability_report object from reliability_report().
palette	One of "web" or "print". See sframe_brand().

Value

A ggplot2 object.

See Also

reliability_report()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("psych", quietly = TRUE)) {
  demo <- sframe_demo_data()
  rr <- reliability_report(demo$responses, demo$instrument, omega = FALSE)
  sframe_plot_reliability(rr)
}
```

---

sframe_plot_sentiment *Sentiment plot: diverging bar, or a positive/negative comparison cloud*

Description

A ggplot2 diverging bar for a `tidy_sentiment` result by default: positive counts extend one direction, negative counts the other, so bar position (not colour alone) carries the primary polarity signal, the same convention `sframe_draw_likert_diverging()` uses for Likert agreement (dark ramp toward the pole) rebuilt here in ggplot2 rather than called directly, since that helper is base-graphics and Likert-scale-specific. Facets by group when `result$table` carries a `group` column, mirroring `sframe_plot_term_frequency()`'s grouped branch.

Usage

```
sframe_plot_sentiment(result, palette = c("web", "print"))
```

Arguments

`result`	A `tidy_sentiment` result list from `run_analysis_plan()`.
`palette`	One of `"web"` or `"print"`. See `sframe_brand()`.

Details

When `result$options$wordcloud` is `TRUE` (opt-in, default `FALSE`, matching `sframe_plot_term_frequency()`'s own word-cloud toggle), draws a comparison cloud instead: negative-sentiment words above the centre line, positive-sentiment words below it, each word sized by how often it occurred, using the internal `tidy_sentiment` runner's `$word_sentiment` word-by-sentiment counts. Answers a different question from the diverging bar: not "how many responses leaned positive," but "which *words* drove that."

Value

A ggplot2 object, or `NULL` when the result carries no table.

See Also

`run_analysis_plan()`, `sframe_draw_likert_diverging()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("tidytext", quietly = TRUE)) {
  demo <- sframe_demo("open_text")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_sentiment(res$RQ7)
}
```

---

sframe_plot_term_frequency

Term-frequency plot: horizontal bar or word cloud

Description

Top terms from a `term_freq` result as a horizontal bar chart, or a word cloud when `result$options$wordcloud` is `TRUE` (opt-in, default `FALSE`). Facets by group when the result carries a `group` role (todo_text_analysis.md section 1a).

Usage

```
sframe_plot_term_frequency(result, palette = c("web", "print"))
```

Arguments

`result`	A `term_freq` result list from `run_analysis_plan()`.
`palette`	One of `"web"` or `"print"`. See `sframe_brand()`.

Value

A ggplot2 object, or `NULL` when the result carries no table.

See Also

`run_analysis_plan()`, `term_frequency()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo("open_text")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_term_frequency(res$RQ1)
}
```

`sframe_plot_topics` *Topic-model top-terms plot: faceted bars, one facet per topic*

Description

Serves both `sframe_run_topic_model_lda()` and `sframe_run_stm_topics()` results with no dispatch on `result$test`: both runners emit a `$table` with the same `topic`/`term`/`beta` columns (LDA's beta from `tidytext::tidy()`, STM's from its fitted word-topic distribution), so this function reads that shared shape directly.

Usage

```
sframe_plot_topics(result, palette = c("web", "print"))
```

Arguments

`result`	A `topic_model_lda` or `stm_topics` result list from `run_analysis_plan()`.
`palette`	One of `"web"` or `"print"`. See `sframe_brand()`.

Value

A ggplot2 object, or `NULL` when the result carries no usable table.

See Also

`sframe_run_topic_model_lda()`, `sframe_run_stm_topics()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("stm", quietly = TRUE) &&
    requireNamespace("tidytext", quietly = TRUE)) {
  demo <- sframe_demo("open_text")
  res <- run_analysis_plan(demo$responses, demo$instrument)
  sframe_plot_topics(res$RQ10)
}
```

sframe_plot_validity *Validity report plot: composite reliability and AVE by construct*

Description

Validity report plot: composite reliability and AVE by construct

Usage

```
sframe_plot_validity(x, palette = c("web", "print"))
```

Arguments

x	An `sframe_validity_report` object from `validity_report()`.
palette	One of `"web"` or `"print"`. See `sframe_brand()`.

Value

A ggplot2 object.

See Also

`validity_report()`

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  loadings <- list(
    sq  = c(sq_1 = 0.80, sq_2 = 0.75, sq_3 = 0.78),
    sat = c(sat_1 = 0.85, sat_2 = 0.82)
  )
  vr <- validity_report(loadings)
  sframe_plot_validity(vr)
}
```

sframe_plot_variable_distribution *Raw-variable distribution panels: histogram, boxplot, and Q-Q*

Description

Unlike `sframe_plot_descriptives()`, which summarises skewness and kurtosis *across* the variables in a `descriptives_report()` table, this operates on one variable's raw values directly (the report table only stores summary statistics, not the underlying vector), matching the pattern `sframe_plot_correlation_matrix()` already uses for report-independent, data-driven plots.

Usage

```
sframe_plot_variable_distribution(data, variable, palette = c("web", "print"))
```

Arguments

data	A data frame of survey responses.
variable	Character. Column name of the variable to plot.
palette	One of "web" or "print". See sframe_brand().

Value

A named list of three ggplot2 objects (histogram, boxplot, qq), or NULL if fewer than two complete values remain.

See Also

descriptives_report(), sframe_plot_descriptives()

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  demo <- sframe_demo_data()
  panels <- sframe_plot_variable_distribution(demo$responses, "sat_1")
  panels$histogram
}
```

---

sframe_plots *Choosing a plot*

Description

surveyframe draws in 3 ways, and which one fits depends on what you already hold.

From a report or a set of results

Every report that has a natural chart carries a plot() method, so a report draws without naming a helper:

- plot(reliability_report(...)) draws one report.
- plot(results) draws each block of an analysis that has a chart.
- plot(results, which = "RQ1") selects one block by its ID.

This is the shortest route, and it is the one to reach for first.

From a helper, by what it takes

The helpers exist for assembling a custom report, where you need one chart on its own terms. They differ in what they accept and in what comes back.

- **A report object, giving 1 plot:**
  - sframe_plot_reliability()
  - sframe_plot_validity()
  - sframe_plot_quality()
  - sframe_plot_missingness()
  - sframe_plot_efa_scree()
  - sframe_plot_efa_loadings()
- **Responses with the instrument, giving 1 plot:**
  - sframe_plot_item_chart()
  - sframe_plot_scale_chart()
  - sframe_plot_likert_matrix()
  - sframe_plot_likert_scale()
  - sframe_plot_correlation_matrix()
  - sframe_plot_descriptives()
- **One block's result, giving 1 plot:**
  - sframe_plot_group_comparison()
  - sframe_plot_paired_comparison()
  - sframe_plot_decision_ranking()
  - sframe_plot_dematel_influence()
- **A regression result, giving 4 panels as a list:**
  - sframe_plot_regression_diagnostics()
- **Responses and 1 column, giving 3 panels as a list:**
  - sframe_plot_variable_distribution()
- **A text result, giving 1 plot:**
  - sframe_plot_term_frequency()
  - sframe_plot_ngram_frequency()
  - sframe_plot_cooccurrence()
  - sframe_plot_cooccurrence_network()
  - sframe_plot_sentiment()
  - sframe_plot_topics()

A helper that finds nothing to draw returns NULL, so guard the result where a report has to keep rendering.

Integration helpers

sframe_draw_mosaic() and sframe_draw_likert_diverging() draw into an open device for the generated reports, and stay exported so those reports keep working. sframe_likert_scale_groups() finds the scales whose items share a choice set, which is how a report groups them. Take the results they draw from run_analysis_plan().

See Also

run_analysis_plan(), render_report()

---

sframe_rated_matrix *Build a performance matrix from rated matrix items*

Description

The third collection path: respondents rate every alternative on every criterion with ordinary matrix items, one item per criterion with the alternatives as its rows, and the decision matrix is the per-cell aggregate. No new item type is needed. Per-cell counts and standard deviations are kept so the report can show how firm each cell is.

Usage

```
sframe_rated_matrix(data, instrument, items, statistic = c("mean", "median"))
```

Arguments

data	A data frame of responses.
instrument	An sframe instrument declaring every id in items.
items	Character vector of "matrix" item ids, one per criterion, in the intended criterion order. Every item must declare the same matrix_items (the alternatives) in the same order. When a decision block also collects weights through a weights_item whose criterion names differ from these ids, the items pair with its criteria in this order, and the result notes each pairing.
statistic	"mean" or "median".

Value

A list with matrix (alternatives x criteria, with dimnames), n, sd, alternatives, criteria, and statistic.

See Also

sframe_collected_weights()

Examples

```
q5    <- sf_choices("q5", 1:5,
           c("Very poor", "Poor", "Fair", "Good", "Excellent"))
price <- sf_item("rate_price", "Rate each supplier: value",
                 type = "matrix", matrix_items = c("Alpha", "Basilica"),
                 choice_set = "q5")
study <- sf_instrument("Supplier selection", components = list(q5, price))
responses <- data.frame(rate_price__Alpha = c(3, 4), rate_price__Basilica = c(5, 5))
rm <- sframe_rated_matrix(responses, study, "rate_price")
rm$matrix
```

sframe_references *List an instrument's references*

Description

List an instrument's references

Usage

```
sframe_references(instrument, cited_by = NULL)
```

Arguments

instrument	An sframe object.
cited_by	Optional target, for example "scale:satisfaction", to list only the references cited there.

Value

A list of sf_reference objects, named by id.

Examples

```
instr <- sframe_demo("likert_scale")$instrument
sframe_references(instr)
```

sframe_reports *Choosing a report*

Description

The 12 report functions split into 2 shapes, and the split is what tells them apart.

Data in, object out

These compute and return an object you can read, coerce with as.data.frame() and often plot.

Function	Answers
codebook_report()	what the instrument declares, as tables
descriptives_report()	the distribution of each variable
missing_data_report()	what is missing, and in what pattern
quality_report()	which responses the declared checks flagged
reliability_report()	alpha and omega per scale
item_report()	how each item behaves inside its scale
efa_report()	whether the data suit a factor analysis

validity_report()	convergent and discriminant validity
assumption_report()	whether a planned test's assumptions hold
posthoc_report()	which pairs differ after an omnibus test

Each returns its primary table through as.data.frame(), with the rest reachable through the accessors in sf_accessors. A failure is carried in the object as an error field, so a report keeps rendering around it.

Object or data in, file out

Function	Writes
render_report()	a whole document, computing its sections from the data
render_results()	a document from analysis results you already have

Reach for render_results() where run_analysis_plan() has already run, and render_report() to go from responses to a document in one call.

See Also

run_analysis_plan(), sframe_plots

---

sframe_result_supplement

The second table a result carries, if it has one

Description

Some results hold a table beside their main one that a reader needs: a quanteda result's leading features, a moderation's conditional slopes at the moderator's own values, or the counts a descriptives block gives for the choice items it names. Both report engines render whatever this returns, under its own caption, so the 2 cannot drift on what a result shows.

Usage

```
sframe_result_supplement(result)
```

Arguments

result	One analysis block's result from run_analysis_plan().

Details

This is exported because the Quarto report template runs in a separate R session against the installed package, so everything it calls has to be part of the public surface. A template calling an internal through ::: fails there and the renderer falls back to the built-in HTML engine without saying why.

Value

A list with `table` and `caption`, or `NULL` where the result has no second table.

See Also

`run_analysis_plan()`, `render_report()`, `analysis_syntax()`

Examples

```
instr <- sf_instrument("Moderation demo", components = list(
  sf_item("y", "Outcome", type = "numeric"),
  sf_item("x", "Predictor", type = "numeric"),
  sf_item("w", "Moderator", type = "numeric")
))
sf_plan(instr) <- list(list(
  id = "RQ1", research_question = "Does w moderate x?",
  family = "inferential", method = "moderation",
  roles = list(outcome = "y", predictor = "x", moderator = "w")
))

set.seed(1)
n <- 80
responses <- data.frame(x = rnorm(n), w = rnorm(n))
responses$y <- 0.4 * responses$x + 0.3 * responses$w +
  0.35 * responses$x * responses$w + rnorm(n, sd = 0.6)
results <- run_analysis_plan(responses, instr)

# the conditional slopes, at the moderator's own values
sframe_result_supplement(results$RQ1)
```

---

sframe_save_figures *Save the charts from an analysis plan as files*

Description

Writes each chart `run_analysis_plan()` drew with `plots = TRUE` to its own file, at the size, resolution and format a journal asks for. The charts the exported `sframe_plot_*()` functions return are `ggplot` objects, so any one of them can also be saved at any size with `ggplot2::ggsave()`.

Usage

```
sframe_save_figures(
  results,
  dir,
  width = 7,
  height = 4.5,
  dpi = 300,
  format = c("png", "tiff", "jpeg", "pdf", "svg")
)
```

Arguments

results	The list run_analysis_plan() returns, run with plots = TRUE.
dir	Directory to write into. Created when absent.
width, height	Size in inches. Default 7 by 4.5.
dpi	Resolution in dots per inch, for the raster formats. Default 300.
format	One of "png" (default), "tiff", "jpeg", "pdf" or "svg".

Value

The paths written, named by block id, invisibly. Only blocks that drew a chart write a file.

See Also

render_report(), which takes the same settings as figures.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  d <- sframe_demo("two_group")
  res <- run_analysis_plan(d$responses, d$instrument, plots = TRUE)
  paths <- sframe_save_figures(res, file.path(tempdir(), "figures"),
                               width = 6, height = 4, dpi = 300)
  basename(paths)
}
```

---

sframe_subset *Subset a surveyframe report*

Description

Keeps the report class, so a subset still prints as a report and still answers as.data.frame().

Value

An object of the same class as x.

`sframe_validation` *Report on a validation result*

Description

`sframe_validation` is the diagnostic object returned by `validate_sframe()` and `validate_model()`. It records whether the object passed, every problem found, and every check that ran, including the checks that found nothing.

Details

Use `sf_is_valid()` for the pass or fail flag, `sf_problems()` for the messages, `as.data.frame()` for a problem-per-row table, `summary()` for the full check roster, and `as_sframe()` to recover the validated instrument.

Value

`print()` returns `x` invisibly. `format()` returns a single character string. `summary()` returns the check table as a data frame. `as.data.frame()` returns one row per problem.

See Also

`validate_sframe()`, `validate_model()`, `sf_problems()`, `sf_is_valid()`, `as_sframe()`

Examples

```
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
item  <- sf_item("sat_1", "The service met my expectations.",
                 type = "likert", choice_set = "ag5", scale_id = "sat")
scale <- sf_scale("sat", "Satisfaction", items = "sat_1")
instr <- sf_instrument("Demo Survey", components = list(cs, item, scale))

v <- validate_sframe(instr, strict = FALSE)
v
sf_is_valid(v)
sf_problems(v)
as.data.frame(v)
summary(v)
```

summary.sf_branch *Summarise an sf_branch object*

Description

Summarise an sf_branch object

Usage

```
## S3 method for class 'sf_branch'
summary(object, ...)
```

Arguments

object	An object of class sf_branch.
...	Ignored. Present for S3 consistency.

Value

object, invisibly.

summary.sf_check *Summarise an sf_check object*

Description

Summarise an sf_check object

Usage

```
## S3 method for class 'sf_check'
summary(object, ...)
```

Arguments

object	An object of class sf_check.
...	Ignored. Present for S3 consistency.

Value

object, invisibly.

summary.sf_choices *Summarise an sf_choices object*

Description

Summarise an sf_choices object

Usage

```
## S3 method for class 'sf_choices'
summary(object, ...)
```

Arguments

object	An object of class sf_choices.
...	Ignored. Present for S3 consistency.

Value

object, invisibly.

summary.sf_item *Summarise an sf_item object*

Description

Summarise an sf_item object

Usage

```
## S3 method for class 'sf_item'
summary(object, ...)
```

Arguments

object	An object of class sf_item.
...	Ignored. Present for S3 consistency.

Value

object, invisibly.

summary.sf_model *Summarise an sf_model object*

Description

Summarise an sf_model object

Usage

```
## S3 method for class 'sf_model'
summary(object, ...)
```

Arguments

object	An object of class sf_model.
...	Ignored. Present for S3 consistency.

Value

object, invisibly.

summary.sf_scale *Summarise an sf_scale object*

Description

Summarise an sf_scale object

Usage

```
## S3 method for class 'sf_scale'
summary(object, ...)
```

Arguments

object	An object of class sf_scale.
...	Ignored. Present for S3 consistency.

Value

object, invisibly.

summary.sframe *Summarise an sframe instrument object*

Description

Prints a structured summary of an `sframe` object including metadata, item type counts, scale definitions, branching rules, and check specifications.

Usage

```
## S3 method for class 'sframe'
summary(object, ...)
```

Arguments

`object`	An object of class `sframe`.
`...`	Ignored. Present for S3 consistency.

Value

`object`, invisibly.

Examples

```
item <- sf_item("q1", "How satisfied are you?", type = "likert",
                choice_set = "agree5")
instr <- sf_instrument("My Survey", components = list(item))
summary(instr)
```

survey_module_server *Shiny module server for an embedded survey*

Description

Draws the survey and collects the respondent's answers. Returns a `reactive` holding `NULL` until the survey has been submitted and saved.

Usage

```
survey_module_server(id, instrument, on_submit = NULL)
```

Arguments

`id`	A character string matching the `id` passed to `survey_module_ui()`.
`instrument`	An `sframe` object, or a `reactive` that returns one.
`on_submit`	Optional function of one argument, called with the response list before the survey is marked complete. Use it to store the response. An error it raises is shown to the respondent, and the survey stays open for another attempt.

Value

A reactive that returns NULL until a response is submitted and on_submit, when supplied, has returned. After that it returns the response list.

Supported item types

Every item type is supported, with the same controls render_survey() uses: likert, single choice, multiple choice, numeric, text, text area, date, slider, rating, ranking, matrix, pairwise comparison and criteria weight, plus section breaks and text blocks.

What is submitted

The response is a named list. It starts with response_id, started_at and submitted_at, followed by one element per response column, named as read_responses() expects. A multi-column item contributes one element per column: item__row for a matrix, item__option for ranking and multiple choice, and one element per pair or criterion for decision items. Values are character.

An item hidden by branching is NA, so an answer given before branching hid its item is never submitted. An unanswered item is NA too. A slider counts as answered once the respondent moves it, and a ranking once the respondent reorders it or chooses "Keep this order". Date questions start empty.

Saving, and a failed save

on_submit is called with the response before the survey is marked complete. If it raises an error, the respondent sees a message and stays on the last page, can submit again, and the returned reactive stays NULL. The thank-you screen appears only after on_submit returns. Without on_submit the module stores nothing, and its thank-you screen tells the respondent the response has not been saved.

Changing the instrument

When instrument is a reactive and its value changes, the survey returns to the welcome screen, the returned reactive goes back to NULL, and no answer given to the previous instrument carries into the new one.

See Also

survey_module_ui(), render_survey(), read_responses()

Examples

```
# survey_module_ui() has a complete example, including on_submit.
```

survey_module_ui *Shiny module UI for an embedded survey*

Description

Places a survey inside a larger Shiny application. Pair with survey_module_server() in the server function. The module shows a welcome screen, the instrument's pages with branching and required-item checks, and a thank-you screen.

Usage

```
survey_module_ui(id, width = "100%")
```

Arguments

id	A character string. The module namespace ID, passed identically to survey_module_server().
width	Character. CSS width for the survey card. Defaults to "100%".

Details

Every item type is drawn with the same controls render_survey() uses, so a response collected through the module has the same columns as one collected there. survey_module_server() describes what is returned.

Value

A shiny.tag object.

See Also

survey_module_server(), render_survey(), export_static_survey()

Examples

```
## Not run:
library(shiny)
library(surveyframe)

cs    <- sf_choices("ag5", 1:5, c("SD", "D", "N", "A", "SA"))
item  <- sf_item("q1", "Rate your experience.", type = "likert",
                 choice_set = "ag5", required = TRUE)
instr <- sf_instrument("Quick Survey", components = list(cs, item))
store <- file.path(tempdir(), "responses.csv")

ui <- fluidPage(
  survey_module_ui("demo"),
  verbatimTextOutput("result")
)
```

```
server <- function(input, output, session) {
  resp <- survey_module_server(
    "demo", instrument = instr,
    # Called before the survey is marked complete. An error here keeps the
    # respondent on the last page with a message, so nothing is lost.
    on_submit = function(response) {
      row <- as.data.frame(response, check.names = FALSE)
      utils::write.table(row, store, sep = ",", row.names = FALSE,
                         col.names = !file.exists(store),
                         append = file.exists(store))
    }
  )
  output$result <- renderPrint({
    req(resp())
    resp()
  })
}

shinyApp(ui, server)

## End(Not run)
```

`term_context` *Keyword-in-context concordance for open-ended text*

Description

Finds every case-insensitive, whole-word match of `term` in `text` and returns the words immediately before and after each match, so a reader can judge how a term is actually being used rather than reading a bare frequency count. Matching is on whole words only, so searching for "room" does not match "roomy". `text` is expected to already have been run through `clean_text_responses()` (its `respondent` attribute is used to cite the original row index for each match; when absent, positions `seq_along(text)` are used instead). Tokenisation here is a plain whitespace split, not the internal stop-word-stripping tokeniser behind `term_frequency()`: stop words are part of a match's context and stripping them would corrupt the very thing a concordance is for.

Usage

```
term_context(text, term, window = 6L, max_matches = 20L)
```

Arguments

`text`	Character vector of responses, ideally already cleaned by `clean_text_responses()`.
`term`	Character. A single keyword to search for.
`window`	Integer. Maximum number of words of context to keep before and after each match. Default `6`.
`max_matches`	Integer. Maximum number of matches to return, counted across all responses. Default `20`.

Value

A data.frame with columns `respondent`, `before`, `match`, and `after`.

Examples

```
demo <- sframe_demo_data()
cleaned <- clean_text_responses(demo$responses, "comments")
term_context(cleaned, "service", window = 4)
```

`term_frequency` *Term frequency for open-ended text*

Description

Tokenises `text` (splitting on whitespace, lower-casing, and stripping punctuation), removes stop words, and counts term frequency. Ships a small built-in English stop-word list so this works with zero optional packages; pass `stop_words = character(0)` to disable filtering, or a custom vector to override it.

Usage

```
term_frequency(text, stop_words = NULL, top_n = 30L)
```

Arguments

`text`	Character vector of responses (raw or already cleaned by `clean_text_responses()`).
`stop_words`	Character vector of words to exclude, or `NULL` to use the built-in English list, or `character(0)` for no filtering.
`top_n`	Integer. Maximum number of terms to return, most frequent first. Default `30`.

Value

A data.frame with columns `term`, `n`, and `pct`.

Examples

```
demo <- sframe_demo_data()
cleaned <- clean_text_responses(demo$responses, "comments")
head(term_frequency(cleaned, top_n = 10))
```

`theme_surveyframe` *surveyframe brand theme for ggplot2*

Description

A `theme_classic()`-based ggplot2 theme (visible axis lines, no floating panel), verified against WCAG 2.2 contrast minimums: 4.5:1 for text, 3:1 for non-text graphical objects. Apply it to any ggplot object, including the plots returned by `run_analysis_plan()` when `plots = TRUE`.

Usage

```
theme_surveyframe(
  base_size = 12,
  base_family = "",
  palette = c("web", "print")
)
```

Arguments

`base_size`	Numeric. Base font size in points. Defaults to 12.
`base_family`	Character. Base font family. Defaults to `""` (the device default).
`palette`	One of `"web"` (brand colours, for on-screen use) or `"print"` (black/grey/white only, for journal-ready figures). See `sframe_brand()` for the verified contrast ratios behind each.

Value

A ggplot2 theme object.

See Also

`run_analysis_plan()`

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) +
  geom_point(colour = "#0E9694") +
  theme_surveyframe()
```

validate_model *Validate a surveyframe model specification*

Description

Checks model IDs, construct IDs, indicators, structural path endpoints, duplicate paths, indirect paths, and engine/type compatibility.

Usage

```
validate_model(model, instrument = NULL, strict = TRUE)
```

Arguments

model	An sf_model() object or compatible list.
instrument	Optional sframe object. When supplied, model indicators must match instrument item IDs.
strict	Logical. When TRUE, invalid models raise an error. When FALSE, problems are reported in the returned diagnostic without stopping.

Value

An sframe_validation object. The model is carried inside it and can be recovered with sf_object().

Changed in 0.4.0

Earlier versions returned the model invisibly when strict = TRUE and a bare unclassed list when strict = FALSE. Both paths now return an sframe_validation object, visibly, so the diagnostic is readable at the console. Code that read $valid and $problems keeps working.

See Also

sframe_validation, sf_problems(), sf_is_valid()

Examples

```
demo <- sframe_demo_data()
m <- sf_model(
  "cb1", type = "cb_sem",
  constructs = list(
    sf_construct("sq", items = c("sq_1", "sq_2", "sq_3")),
    sf_construct("sat", items = c("sat_1", "sat_2"))
  ),
  paths = list(sf_path("sq", "sat"))
)
diag <- validate_model(m, instrument = demo$instrument)
diag
```

validate_sframe *Validate an instrument object*

Description

Checks the internal consistency of an sframe instrument object and returns a diagnostic result. Validation is performed automatically by write_sframe() and optionally by read_sframe(). It can also be run independently at any point during instrument construction.

Usage

```
validate_sframe(instrument, strict = TRUE)
```

Arguments

instrument	An sframe object created by sf_instrument().
strict	Logical. When TRUE (default), any detected problem raises an error of class sframe_validation_error. When FALSE, problems are reported in the returned diagnostic without stopping.

Details

The following checks are performed:

- Duplicate item IDs
- Invalid item IDs
- Duplicate choice-set IDs
- Duplicate scale IDs
- Item IDs, scale IDs and response columns that share a name, or take a reserved metadata name such as submitted_at. A scale's score is stored in a column named by its ID, so a shared name would overwrite data
- Items with missing labels
- Items referencing a missing choice_set in the instrument
- Items referencing a missing scale_id in the instrument
- Items marked reverse = TRUE without a scale_id
- Choice sets referenced by items but not present in the instrument
- Scale items vectors containing IDs not present in the instrument
- Scale scoring parameters: repeated items, a min_valid outside 1 to the number of items, and weights that are not positive finite numbers
- Reverse coding declared for an item outside the scale that declares it
- Branching rules referencing item IDs not present in the instrument
- %in% branching rules whose value no evaluator can consume
- Attention checks referencing item IDs not present in the instrument
- Analysis plan roles referencing missing variables or models
- Model specifications referencing missing indicators or constructs

Value

An sframe_validation object. When the instrument is valid, the instrument carried inside it has `meta$validated` set to `TRUE` and can be recovered with `as_sframe()`.

Changed in 0.4.0

Earlier versions returned two different things depending on `strict`: the instrument itself, invisibly, when `strict = TRUE`, and a bare unclassed list when `strict = FALSE`. A validator should report a diagnostic, so both paths now return an sframe_validation object, and they return it visibly, so `validate_sframe(instrument)` typed at the console shows the result. Code that read `$valid` and `$problems` keeps working. Code that used the `strict = TRUE` return as an instrument should now wrap the call in `as_sframe()`.

See Also

sframe_validation, `as_sframe()`, `sf_problems()`, `sf_is_valid()`, `sf_instrument()`, `write_sframe()`

Examples

```
# Build a minimal valid instrument and validate it
cs    <- sf_choices("ag5", 1:5,
           c("Strongly disagree", "Disagree", "Neutral",
             "Agree", "Strongly agree"))
item  <- sf_item("sat_1", "The service met my expectations.",
                 type = "likert", choice_set = "ag5", scale_id = "sat")
scale <- sf_scale("sat", "Satisfaction", items = "sat_1")
instr <- sf_instrument("Demo Survey", components = list(cs, item, scale))

# The result prints its own diagnostic
validate_sframe(instr, strict = FALSE)

# Explore it with dedicated methods rather than reaching in with `$`
v <- validate_sframe(instr, strict = FALSE)
sf_is_valid(v)
sf_problems(v)
summary(v)

# Recover the validated instrument
validated <- as_sframe(validate_sframe(instr, strict = TRUE))
isTRUE(sf_meta(validated)$validated)
```

`validity_report` *Validity report for construct models*

Description

Validity report for construct models

Usage

```
validity_report(loadings, construct_scores = NULL, items_by_construct = NULL)
```

Arguments

loadings	A data.frame with columns construct, item, and loading, or a named list of loading vectors by construct.
construct_scores	Optional data.frame of construct scores for Fornell-Larcker and inter-construct correlations.
items_by_construct	Optional named list, one element per construct, each a data.frame of that construct's item-level responses. When supplied, htmt is the Henseler heterotrait-monotrait ratio: the mean absolute heterotrait-heteromethod correlation over the geometric mean of the two constructs' mean absolute monotrait-heteromethod correlations. Constructs with a single item have no monotrait correlations, so their HTMT entries are NA. Without this argument, htmt falls back to the absolute inter-construct correlation matrix from construct_scores (the pre-0.3.4 behaviour). The htmt_method element records which was computed.

Value

An object of class sframe_validity_report.

Examples

```
loadings <- list(
  sq  = c(sq_1 = 0.80, sq_2 = 0.75, sq_3 = 0.78),
  sat = c(sat_1 = 0.85, sat_2 = 0.82)
)
vr <- validity_report(loadings)
vr$reliability
```

---

write_sframe *Write an instrument to a .sframe file*

Description

Serialises an sframe instrument object to a UTF-8 JSON file with a SHA-256 integrity hash. The instrument is always validated before writing, and an invalid instrument is refused. The hash is computed over a canonical serialisation of the content with hash.value set to an empty string: object keys are sorted, so it identifies content, not the exact bytes.

Usage

```
write_sframe(
  instrument,
  path,
  pretty = TRUE,
  overwrite = FALSE,
  new_instrument = FALSE
)
```

Arguments

instrument	An sframe object created by sf_instrument().
path	Character. The file path to write to. The .sframe extension is appended automatically if not already present.
pretty	Logical. Whether to write formatted JSON with indentation. Defaults to TRUE. Set to FALSE for compact files.
overwrite	Logical. Whether to overwrite an existing file. Defaults to FALSE.
new_instrument	Logical. TRUE declares that the content is a new instrument, not a revision of the one it was read from, so no amendment is required. The instrument must carry no amendment log, and an instrument read with read_sframe() must have a new meta$title or meta$version, so 2 different files never claim to be one instrument. Defaults to FALSE.

Value

The file path, invisibly.

Revisions and the amendment log

Writing checks the amendment log recorded by amend_sframe(). Each entry must follow the one before it, the instrument must still match its last recorded amendment, and an instrument read with read_sframe() must keep every amendment it was read with. Content changed since it was read is refused unless the change was recorded with amend_sframe(). To publish changed content as a different instrument, remove its amendment log and set new_instrument = TRUE.

These are checks on the content in hand. The hash is unsigned and can be recomputed by anyone who edits a file, so it does not establish who wrote an instrument or when.

See Also

read_sframe(), validate_sframe()

Examples

```
instr <- read_sframe(
  system.file("extdata", "tourism_services_demo.sframe",
              package = "surveyframe")
)
out <- write_sframe(instr, tempfile(fileext = ".sframe"))
file.exists(out)
```

Index

add_model, 8
add_references, 8
add_references(), *75*, *114, 115*
amend_sframe, 9
amend_sframe(), *9*, *11, 12*, *54*, *71*, *176*
amendment_log, 11
amendment_log(), *11*
analysis_syntax, 12
analysis_syntax(), *75*, *160*
as.data.frame(), *158, 159*
as.data.frame.sf_choices, 14
as.data.frame.sframe, 14
as.data.frame.sframe_analysis_results, 15
as.data.frame.sframe_assumption_report, 16
as.data.frame.sframe_codebook, 16
as.data.frame.sframe_descriptives_report, 17
as.data.frame.sframe_efa_report, 18
as.data.frame.sframe_efa_solution, 18
as.data.frame.sframe_item_report, 19
as.data.frame.sframe_missing_data_report, 20
as.data.frame.sframe_quality_report, 20
as.data.frame.sframe_reliability_report, 21
as.data.frame.sframe_sample_size_plan, 22
as.data.frame.sframe_sensitivity, 22
as.data.frame.sframe_validation, 23
as.data.frame.sframe_validity_report, 24
as_sframe, 24
as_sframe(), *6*, *89*, *111*, *119*, *162*, *174*
assumption_report, 25
assumption_report(), *16*, *159*

base::as.data.frame(), *14–24*, *122*
bootstrap_ci, 26
bootstrap_ci(), *7*, *32*, *37*

cfa_lavaan_syntax, 27
cfa_lavaan_syntax(), *28*
cfa_syntax, 28
cfa_syntax(), *6*, *13*, *27*, *34*, *116*
clean_text_responses, 29
clean_text_responses(), *42*, *57*, *169, 170*
codebook_report, 30
codebook_report(), *7*, *15*, *17*, *76*, *89*, *121, 122*, *127*, *158*
cohens_d_ci, 31
cohens_d_ci(), *7*, *26*
cramers_v_ci, 32
cramers_v_ci(), *7*, *26*

descriptives_report, 33
descriptives_report(), *6*, *17*, *141*, *154, 155*, *158*

efa_report, 34
efa_report(), *6*, *18*, *28*, *143*, *158*
efa_solution, 35
efa_solution(), *19*, *34*, *142*
efa_syntax, 36
efa_syntax(), *27*
eta_sq_ci, 37
eta_sq_ci(), *7*, *26*
export_google_sheet, 38
export_google_sheet(), *72*
export_static_survey, 39
export_static_survey(), *48*, *60, 61*, *168*
extract_quotes, 42

format.sf_branch, 43
format.sf_check, 43
format.sf_choices, 44
format.sf_item, 44
format.sf_model, 45

format.sf_scale, 45
format.sframe, 46

ggplot2::ggsave(), *160*

interactive(), *60*
item_report, 46
item_report(), *6*, *19*, *73*, *116*, *158*

launch_builder, 47
launch_builder(), *6*, *41*, *49*, *52*
launch_builder_demo, 48
launch_builder_demo(), *135*
launch_dashboard, 49
launch_dashboard(), *52*
launch_dashboard_demo, 51
launch_dashboard_demo(), *49, 50*, *135*
launch_studio, 51
launch_studio(), *41*, *48–50*, *60, 61*, *80*
launch_studio_demo, 53
launch_studio_demo(), *135*
link_git_commit, 54

missing_data_report, 55
missing_data_report(), *6*, *20*, *147*, *158*
model_json, 55
model_report_template, 56

ngram_frequency, 57
ngram_frequency(), *147*

outlier_report, 58

pagedown::chrome_print(), *75*
plot.sframe_analysis_results, 59
posthoc_report, 59
posthoc_report(), *159*
preview_demo, 60
preview_demo(), *131*
print.sf_branch, 61
print.sf_check, 62
print.sf_choices, 62
print.sf_item, 63
print.sf_model, 64
print.sf_scale, 64
print.sframe, 65

quality_report, 65
quality_report(), *6*, *21*, *50*, *70*, *72*, *76*, *92, 93*, *100*, *108, 109*, *149*, *158*

read_bibtex, 67
read_bibtex(), *9*, *115*
read_responses, 68
read_responses(), *6*, *39*, *49*, *52*, *66, 67*, *72*, *80, 81*, *120, 121*, *123*, *134*, *145*, *167*
read_sframe, 70
read_sframe(), *10, 11*, *47, 48*, *52*, *126*, *173*, *176*
read_sheet_responses, 72
read_sheet_responses(), *6*, *38, 39*, *49*, *81, 82*
reliability_report, 73
reliability_report(), *6*, *21*, *28*, *34*, *47*, *72*, *76*, *85*, *116, 117*, *150*, *158*
render_report, 74
render_report(), *7*, *13*, *31*, *77–79*, *114*, *123*, *157*, *159–161*
render_result, 77
render_results, 78
render_results(), *7*, *82*, *159*
render_survey, 80
render_survey(), *6*, *41*, *102*, *167, 168*
run_analysis_plan, 81
run_analysis_plan(), *6, 7*, *12, 13*, *15*, *48*, *50*, *59*, *77–79*, *88*, *90*, *113*, *124*, *129*, *137–140*, *144*, *147–153*, *156, 157*, *159–161*, *171*

sample_size_plan, 83
sample_size_plan(), *22*
score_scales, 84
score_scales(), *6, 7*, *47*, *50*, *67*, *70*, *72*, *116, 117*
sem_lavaan_syntax, 85
sem_lavaan_syntax(), *27*
seminr_syntax, 86
seminr_syntax(), *27*
sensitivity_analysis, 87
sf_accessors, 88, *91*, *93, 94*, *107, 108*, *110*, *112*, *118*, *121, 122*, *159*
sf_apa, 89
sf_branch, 90
sf_branch(), *6*, *100*, *102, 103*, *107*
sf_branches, 91
sf_branches(), *96*
sf_check, 92
sf_check(), *6*, *67*, *100*, *102, 103*, *107*
sf_checks, 93
sf_checks(), *96*

sf_choice_sets, 94
sf_choice_sets(), *96*, *122*
sf_choices, 94
sf_choices(), *6*, *14*, *100*, *102*, *103*, *105–107*
sf_component_list, *88*, *89*, *91*, *93*, *94*, 96, *107*, *110*, *118*
sf_conjoint_design, 96
sf_conjoint_design(), *102*
sf_construct, 98
sf_construct(), *109*
sf_covariance, 99
sf_covariance(), *27*, *109*
sf_flagged, 99
sf_flagged(), *109*
sf_id, 100
sf_id(), *107*
sf_identity, *100*, 101, *107*
sf_indirect, 101
sf_indirect(), *109*
sf_instrument, 102
sf_instrument(), *6*, *66*, *69*, *91*, *93*, *95*, *97*, *106*, *117*, *131*, *173*, *174*, *176*
sf_is_valid, 104
sf_is_valid(), *162*, *172*, *174*
sf_item, 104
sf_item(), *6*, *7*, *92*, *93*, *95*, *100*, *102*, *103*, *107*, *117*
sf_items, 106
sf_items(), *7*, *15*, *96*, *122*
sf_label, 107
sf_label(), *100*
sf_meta, 108
sf_meta(), *7*
sf_missing_columns, 108
sf_model, 109
sf_model(), *8*, *27*, *28*, *56*, *86*, *100*, *102*, *107*, *172*
sf_models, 110
sf_models(), *7*, *96*
sf_object, 111
sf_object(), *172*
sf_path, 111
sf_path(), *109*
sf_plan, 112
sf_plan(), *7*, *113*
sf_plan<-, *112*, 113
sf_problems, 113
sf_problems(), *23*, *89*, *162*, *172*, *174*
sf_reference, 114
sf_reference(), *8*, *68*, *75*, *102*, *126*
sf_report_accessors, *90*, *100*, 116
sf_scale, 116
sf_scale(), *6*, *7*, *47*, *73*, *85*, *100*, *102*, *103*, *106*, *107*, *136*, *145*, *146*
sf_scales, 118
sf_scales(), *7*, *96*, *122*
sf_validation_accessors, *104*, *111*, *114*, 118
sframe_aggregate_judgements, 119
sframe_aggregate_judgements(), *123*, *128*
sframe_analysis_qmd, 120
sframe_as_data_frame, *14–24*, 121
sframe_assemble_pairwise, 122
sframe_assemble_pairwise(), *119*, *120*, *128*
sframe_bibliography, 123
sframe_builder_empty_state, 124
sframe_builder_state_from_instrument, 125
sframe_builder_validate_draft, 125
sframe_codebook_items_display, 127
sframe_collected_weights, 127
sframe_collected_weights(), *123*, *157*
sframe_decision_options, 128
sframe_decision_options(), *88*
sframe_dematel_compute, 129
sframe_demo, 130
sframe_demo(), *61*, *131*, *133–135*
sframe_demo_branding, 131
sframe_demo_branding(), *130*, *133*
sframe_demo_data, 132
sframe_demo_qmd, 132
sframe_demo_qmd(), *120*, *121*, *133*
sframe_demos, 133
sframe_demos(), *7*, *60*, *120*, *130–133*
sframe_draw_likert_diverging(), *144*, *145*, *151*, *156*
sframe_draw_mosaic(), *156*
sframe_export_labelled, 134
sframe_export_labelled(), *131*
sframe_input_types_demo_data, 135
sframe_input_types_demo_data(), *130*, *131*
sframe_likert_scale_groups, 136
sframe_likert_scale_groups(), *156*
sframe_plot_cooccurrence, 136

sframe_plot_cooccurrence(), *156*
sframe_plot_cooccurrence_network, 137
sframe_plot_cooccurrence_network(), *156*
sframe_plot_correlation_matrix, 138
sframe_plot_correlation_matrix(), *154*, *156*
sframe_plot_decision_ranking, 139
sframe_plot_decision_ranking(), *140*, *156*
sframe_plot_dematel_influence, 140
sframe_plot_dematel_influence(), *156*
sframe_plot_descriptives, 141
sframe_plot_descriptives(), *154–156*
sframe_plot_efa_loadings, 142
sframe_plot_efa_loadings(), *156*
sframe_plot_efa_scree, 143
sframe_plot_efa_scree(), *156*
sframe_plot_group_comparison, 143
sframe_plot_group_comparison(), *156*
sframe_plot_item_chart(), *156*
sframe_plot_likert_matrix, 144
sframe_plot_likert_matrix(), *136*, *146*, *156*
sframe_plot_likert_scale, 145
sframe_plot_likert_scale(), *136*, *156*
sframe_plot_missingness, 146
sframe_plot_missingness(), *156*
sframe_plot_ngram_frequency, 147
sframe_plot_ngram_frequency(), *156*
sframe_plot_paired_comparison, 148
sframe_plot_paired_comparison(), *156*
sframe_plot_quality, 149
sframe_plot_quality(), *156*
sframe_plot_regression_diagnostics, 149
sframe_plot_regression_diagnostics(), *156*
sframe_plot_reliability, 150
sframe_plot_reliability(), *7*, *156*
sframe_plot_scale_chart(), *156*
sframe_plot_sentiment, 151
sframe_plot_sentiment(), *156*
sframe_plot_term_frequency, 152
sframe_plot_term_frequency(), *147*, *151*, *156*
sframe_plot_topics, 153
sframe_plot_topics(), *156*
sframe_plot_validity, 154
sframe_plot_validity(), *156*
sframe_plot_variable_distribution, 154
sframe_plot_variable_distribution(), *156*
sframe_plots, 155, *159*
sframe_rated_matrix, 157
sframe_references, 158
sframe_references(), *115*
sframe_reports, 158
sframe_result_supplement, 159
sframe_result_supplement(), *77*
sframe_run_stm_topics(), *42*, *153*
sframe_run_topic_model_lda(), *153*
sframe_save_figures, 160
sframe_save_figures(), *76*, *78*
sframe_subset, 161
sframe_validation, *23–25*, *89*, *104*, *111*, *114*, *119*, 162, *172*, *174*
shiny::runApp(), *50*, *52*
stats::median(), *26*
stats::power.anova.test(), *84*
stats::power.t.test(), *84*
summary.sf_branch, 163
summary.sf_check, 163
summary.sf_choices, 164
summary.sf_item, 164
summary.sf_model, 165
summary.sf_scale, 165
summary.sframe, 166
survey_module_server, 166
survey_module_server(), *168*
survey_module_ui, 168
survey_module_ui(), *166*, *167*
surveyframe *(*surveyframe-package*)*, 6
surveyframe-package, 6

table(), *32*
tempdir(), *40*
term_context, 169
term_frequency, 170
term_frequency(), *7*, *57*, *137*, *152*, *169*
theme_surveyframe, 171

utils::browseURL(), *48*

validate_model, 172
validate_model(), *162*
validate_sframe, 173

validate_sframe(), *6, 7*, *24, 25*, *71*, *91*, *103, 104*, *113, 114*, *119*, *162*, *176*
validity_report, 174
validity_report(), *24*, *138, 139*, *154*, *159*

write_sframe, 175
write_sframe(), *6*, *10, 11*, *39*, *54*, *71*, *103*, *121, 122*, *126*, *173, 174*